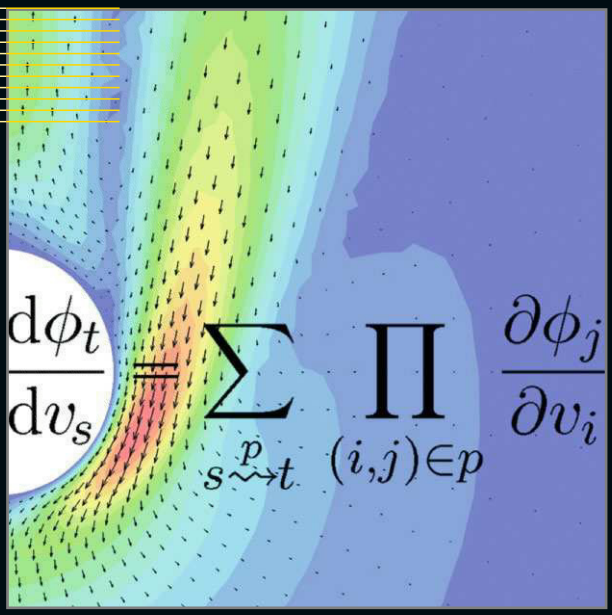
Lecture Notes in Computational Science and Engineering

50



Martin Bücker
George Corliss
Paul Hovland
Uwe Naumann
Boyana Norris
Editors

Automatic Differentiation: Applications, Theory, and Implementations

Springer

Lecture Notes
in Computational Science
and Engineering

50

Editors

Martin Bücker George Corliss Paul Hovland
Uwe Naumann Boyana Norris (Eds.)

Automatic Differentiation: Applications, Theory, and Implementations

With 108 Figures and 33 Tables

Springer

Editors

Martin Bücker
Institute for Scientific Computing
RWTH Aachen University
D-52056 Aachen, Germany
email: buecker@sc.rwth-aachen.de

Uwe Naumann
Software and Tools
for Computational Engineering
RWTH Aachen University
D-52056 Aachen, Germany
email: naumann@stce.rwth-aachen.de

George Corliss
Department of Electrical
and Computer Engineering
Marquette University
1515 W. Wisconsin Avenue
P.O. Box 1881
Milwaukee, WI 53201-1881, USA
email: george.corliss@marquette.edu

Paul Hovland
Boyana Norris
Mathematics and Computer Science Division
Argonne National Laboratory
9700 S Cass Ave
Argonne, IL 60439, USA
email: hovland@mcs.anl.gov
norris@mcs.anl.gov

Library of Congress Control Number: 2005934452

Mathematics Subject Classification: 65Y99, 90C31, 68N19

ISBN-10 3-540-28403-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28403-1 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media
springer.com

Printed in The Netherlands

Typesetting: by the authors and TechBooks using a Springer LaTeX macro package

Cover design: *design & production* GmbH, Heidelberg

Printed on acid-free paper SPIN: 11360537 46/TechBooks 5 4 3 2 1 0

Preface

The Fourth International Conference on Automatic Differentiation was held July 20-23 in Chicago, Illinois. The conference included a one day short course, 42 presentations, and a workshop for tool developers. This gathering of automatic differentiation researchers extended a sequence that began in Breckenridge, Colorado, in 1991 and continued in Santa Fe, New Mexico, in 1996 and Nice, France, in 2000. We invited conference participants and the general automatic differentiation community to submit papers to this special collection. The 28 accepted papers reflect the state of the art in automatic differentiation.

The number of automatic differentiation tools based on compiler technology continues to expand. The papers in this volume discuss the implementation and application of several compiler-based tools for Fortran, including the venerable ADIFOR, an extended NAGWare compiler, TAF, and TAPENADE. While great progress has been made toward robust, compiler-based tools for C/C++, most notably in the form of the ADIC and TAC++ tools, for now operator-overloading tools such as ADOL-C remain the undisputed champions for reverse-mode automatic differentiation of C++. Tools for automatic differentiation of high level languages, including COSY and ADiMat, continue to grow in importance as the productivity gains offered by high-level programming are recognized.

The breadth of automatic differentiation applications also continues to expand. This volume includes papers on accelerator design, chemical engineering, weather and climate modeling, dynamical systems, circuit device modeling, structural dynamics, and radiation treatment planning. The last application is representative of a general trend toward more applications in the biological sciences. This is an important trend for the continued growth of automatic differentiation, as new applications identify novel uses for automatic differentiation and present new challenges to tool developers. The papers in this collection demonstrate both the power of automatic differentiation to facilitate new scientific discoveries and the ways in which application requirements can drive new developments in automatic differentiation.

Advances in automatic differentiation theory take many forms. Progress in mathematical theory expands the scope of automatic differentiation and its variants or identifies new uses for automatic differentiation capabilities. Advances in combinatorial theory reduce the cost of computing or storing derivatives. New compiler theory identifies analyses that reduce the time or storage requirements for automatic differentiation, especially for the reverse mode. This collection includes several papers on mathemetical and combinatorial theory. Furthermore, several of the tools papers document the compiler analyses that are required to construct an effective automatic differentiation tool.

This collection is organized as follows. The first two papers, by Rall and Werbos, provide an overview of automatic differentiation and place it in a historical context. The first section, comprising seven papers, covers advances in automatic differentiation theory. The second section, containing eight papers, describes the implementation of automatic differentiation tools. The final section, devoted to applications, includes eleven papers discussing new uses for automatic differentiation. Many papers include elements of two or more of the general themes of theory, tools, and applications. For example, in many cases successful application of an automatic differentiation tool in a new domain requires advances in theory or tools. A collected bibliography includes all of the references cited by one of the papers in this volume, as well as all of the papers from the proceeedings of the first three conferences. The bibliography was assembled from the BibTeX database at autodiff.org, an emerging portal for the automatic differentiation community. We thank Heiner Bach for his hard work on the autodiff.org bibliographic database.

While the last four years have seen many advances in automatic differentiation theory and implementation, many challenges remain. We hope that the next International Conference on Automatic Differentiation includes reports of reverse mode source transformation tools for templated C++, proofs that minimizing the number of operations in a Jacobian computation is NP-complete, advances in the efficient computation of Hessians, and many examples of new applications that identify research challenges and drive tool development forward. Together, researchers in automatic differentiation applications, theory, and implementation can advance the field in new and unexpected ways.

Martin Bücker
George Corliss
Paul Hovland
Uwe Naumann
Boyana Norris

Contents

List of Contributors

Mauricio Araya-Polo
INRIA Sophia-Antipolis, projet TROPICS
2004 route des Lucioles
BP 93
06902 Sophia-Antipolis
France
Mauricio.Araya@sophia.inria.fr

Paul I. Barton
Massachusetts Institute of Technology
Department of Chemical Engineering
77 Massachusetts Ave.
Cambridge, MA 02139
USA
pib@mit.edu

Martin Berz
Michigan State University
Department of Physics and Astronomy
East Lansing, MI 48824
USA
berz@msu.edu

Christian H. Bischof
Institute for Scientific Computing
RWTH Aachen University
D–52056 Aachen
Germany
bischof@sc.rwth-aachen.de

Frances Bodman
Cranfield University, RMCS Shrivenham
Computer Information Systems Engineering Dept.
Swindon
SN6 8LA
United Kingdom

H. Martin Bücker
Institute for Scientific Computing
RWTH Aachen University
D–52056 Aachen
Germany
buecker@sc.rwth-aachen.de

Richard Casey
Center for Applied Mathematics
Cornell University
Ithaca, NY 14853–2401
USA
rjc20@cornell.edu

William Castaings
Laboratoire de Modélisation et Calcul
Institut d'Informatique et Mathématiques Appliquées de Grenoble
BP 53
38041 Grenoble Cedex 9
France
william.castaings@imag.fr

Bruce Christianson
University of Hertfordshire, Hatfield
Computer Science
College Lane
Hatfield
Hatfield, Herts.
AL10 9AB
United Kingdom
B.Christianson@herts.ac.uk

Carlos E. Christoffersen
Department of Electrical Engineering
Lakehead University
955 Oliver Road
Thunder Bay, Ontario P7B 5E1
Canada
c.christoffersen@ieee.org

George F. Corliss
Electrical and Computer Engineering
Marquette University
P.O. Box 1881
Milwaukee WI 53201–1881
USA
George.Corliss@Marquette.edu

Maurice Cox
National Physical Laboratories
Teddington
TW11 0LW
United Kingdom
Maurice.Cox@npl.co.uk

Denis Dartus
Institut de Mécanique des Fluides de Toulouse
Hydrodynamique et Environnement
1 Allée du Professeur Camille Soula
31400 Toulouse
France
dartus@imft.fr

Ronald Errico
Global Modeling and Assimilation Office, Code 610.1
Goddard Space Flight Center
Greenbelt, MD 20771
USA
rerrico@gmao.gsfc.nasa.gov

Harley Flanders
University of Michigan
1867 East Hall
Ann Arbor, MI 48109–1043
USA
harley@umich.edu

Shaun Forth
Applied Mathematics and Operational Research
Engineering Systems Department
Cranfield University, Shrivenham Campus
Swindon
SN6 8LA
United Kingdom
S.A.Forth@cranfield.ac.uk

Benedick A. Fraass
The University of Michigan Medical School
Department of Radiation Oncology
1500 E. Medical Center Dr.
Ann Arbor, MI 48109–0010
USA

Jidong Gao
Center for Analysis and Prediction of Storms
University of Oklahoma
Sarkeys Energy Center, Suite 1110
100 East Boyd Street
Norman, OK 73019–1011
USA
jdgao@ou.edu

David M. Gay
Sandia National Labs
P.O. Box 5800, MS 0370
Albuquerque, NM 87185–0370
USA
dmgay@sandia.gov

Ronald Gelaro
Global Modeling and Assimilation Office, Code 610.1
Goddard Space Flight Center
Greenbelt, MD 20771
USA
gelaro@gmao.gsfc.nasa.gov

Ralf Giering
FastOpt
Schanzenstr. 36
D–20357 Hamburg
Germany
Rald@FastOpt.com

Andreas Griewank
Humboldt-Universität zu Berlin
Institut für Mathematik
Unter den Linden 6
D–10099 Berlin
Germany
griewank@mathematik.hu-berlin.de

Johannes Grote
Michigan State University
Department of Physics and Astronomy
East Lansing, MI 48824
USA
grotejoh@msu.edu

John Guckenheimer
Mathematics Department
Cornell University
Ithaca, NY 14853–2401
USA
gucken@cam.cornell.edu

Frank P. Hart
Department of Electrical and Computer Engineering
North Carolina State University
P.O. Box 7914
Raleigh, NC 27695–7914
USA
fphart@eos.ncsu.edu

Laurent Hascoët
INRIA Sophia-Antipolis, projet TROPICS
2004 route des Lucioles
BP 93
06902 Sophia-Antipolis
France
Laurent.Hascoet@sophia.inria.fr

Marc Honnorat
Laboratoire de Modélisation et Calcul
Institut d'Informatique et Mathématiques Appliquées de Grenoble
BP 53
38041 Grenoble Cedex 9
France
marc.honnorat@imag.fr

Shahadat Hossain
University of Lethbridge
4401 University Drive
T1K 3M4 Lethbridge, AB
Canada
shahadat.hossain@uleth.ca

Paul D. Hovland
University of Chicago / Argonne National Laboratory
9700 S. Cass Ave
Argonne, IL 60439
USA
hovland@mcs.anl.gov

Kyung-Wook Jee
The University of Michigan Medical School
Department of Radiation Oncology
1500 E. Medical Center Dr.
Ann Arbor, MI 48109–0010
USA
wook@umich.edu

Carol Johnstone
MS 221 Fermilab
Kirk Rd. & Wilson St.
Batavia, IL 60510
USA
cjj@fnal.gov

Thomas Kaminski
FastOpt
Schanzenstr. 36
D–20357 Hamburg
Germany
Thomas@FastOpt.com

Andy J. Keane
University of Southampton
School of Engineering Sciences
Mechanical Engineering
Highfield, Southampton
SO17 1BJ
United Kingdom
ajk@soton.ac.uk

Nikhil Kriplani
Department of Electrical and Computer Engineering
North Carolina State University
P.O. Box 7914
Raleigh, NC 27695–7914
USA
nmkripla@unity.ncsu.edu

Koichi Kubota
Dept. Information and System Engineering
Chuo University
1-13-27 Kasuga, Bunkyo-ku
112-8551 Tokyo
Japan
kubota@ise.chuo-u.ac.jp

François-Xavier Le Dimet
Laboratoire de Modélisation et Calcul
Institut d'Informatique et Mathématiques Appliquées de Grenoble
BP 53
38041 Grenoble Cedex 9
France
francois-xavier.le-dimet@imag.fr

Youssef Loukili
Laboratoire de Modélisation et Calcul
Institut d'Informatique et Mathématiques Appliquées de Grenoble
BP 53
38041 Grenoble Cedex 9
France
youssef.loukili@imag.fr

Sonali R. Luniya
Department of Electrical and Computer Engineering
North Carolina State University
P.O. Box 7914
Raleigh, NC 27695–7914
USA
srluniya@unity.ncsu.edu

Kyoko Makino
Michigan State University
Department of Physics and Astronomy
East Lansing, MI 48824
USA
makino@msu.edu

Shashikant L. Manikonda
Michigan State University
Department of Physics and Astronomy
East Lansing, MI 48823
USA
manikond@msu.edu

Wolfgang Marquardt
Process Systems Engineering
RWTH Aachen University
D–52056 Aachen
Germany
marquardt@lpt.rwth-aachen.de

William Martin
Center for Analysis and Prediction of Storms
University of Oklahoma
Sarkeys Energy Center, Suite 1110
100 East Boyd Street
Norman, OK 73019–1011
USA
wjmartin@ou.edu

Daniel L. McShan
The University of Michigan Medical School
Department of Radiation Oncology
1500 E. Medical Center Dr.
Ann Arbor, MI 48109–0010
USA

Jérôme Monnier
Laboratoire de Modélisation et Calcul
Institut d'Informatique et Mathématiques Appliquées de Grenoble
BP 53
38041 Grenoble Cedex 9
France
jerome.monnier@imag.fr

Uwe Naumann
RWTH Aachen University
LuFG Software and Tools for Computational Engineering
D–52056 Aachen
Germany
naumann@stce.rwth-aachen.de

Boyana Norris
University of Chicago / Argonne National Laboratory
9700 S. Cass Ave
Argonne, IL 60439
USA
norris@mcs.anl.gov

Derya B. Özyurt
Massachusetts Institute of Technology
Department of Chemical Engineering
77 Massachusetts Ave.
Cambridge, MA 02139
USA
derya@mit.edu

Valérie Pascual
INRIA Sophia-Antipolis, projet TROPICS
2004 route des Lucioles
BP 93
06902 Sophia-Antipolis
France
valerie.pascual@sophia.inria.fr

Monika Petera
Institute for Scientific Computing
RWTH Aachen University
D–52056 Aachen
Germany
petera@sc.rwth-aachen.de

Eric T. Phipps
Sandia National Laboratories
P.O. Box 5800 MS-0316
Albuquerque, NM 87185–0370
USA
etphipp@sandia.gov

John D. Pryce
Cranfield University, RMCS Shrivenham
Computer Information Systems Engineering Dept.

Swindon
SN6 8LA
United Kingdom
J.D.Pryce@cranfield.ac.uk

Louis Rall
University of Wisconsin-Madison
5101 Odana Road
Madison, WI 53711
USA
rall@math.wisc.edu

Jan Riehme
Humboldt-Universität zu Berlin
Institut für Mathematik
Unter den Linden 6
D–10099 Berlin
Germany
riehme@mathematik.hu-berlin.de

Sebastian Schlenkrich
Institute for Scientific Computing
Technical University Dresden
D–01062 Dresden
Germany
schlenk@math.tu-dresden.de

Pavel Snopok
MS 221 Fermilab
Kirk Rd. & Wilson St.
Batavia, IL 60510
USA
snopok@fnal.gov

Michael B. Steer
Department of Electrical and
Computer Engineering
North Carolina State University
P.O. Box 7914
Raleigh, NC 27695–7914
USA
m.b.steer@ieee.org

Trond Steihaug
Department of Informatics
University of Bergen
N-5020 Bergen
Norway
trond.steihaug@ii.uib.no

Julia Sternberg
Technical University Dresden
Department of Mathematics
Institute of Scientific Computing
D–01062 Dresden
Germany
jstern@math.tu-dresdent.de

Mohamed Tadjouddine
Applied Mathematics and
Operational Research
Engineering Systems Department
Cranfield University, Shrivenham
Campus
Swindon
SN6 8LA
United Kingdom
M.Tadjouddine@cranfield.ac.uk

Ricardo Todling
Global Modeling and Assimilation
Office, Code 610.1
Goddard Space Flight Center
Greenbelt, MD 20771
USA
rtodling@gmao.gsfc.nasa.gov

Jean Utke
University of Chicago / Argonne
National Laboratory
9700 S. Cass Ave
Argonne, IL 60439
USA
utke@mcs.anl.gov

Andre Vehreschild
Institute for Scientific Computing
RWTH Aachen University
D–52056 Aachen
Germany
vehreschild@sc.rwth-aachen.de

Michael Voßbeck
FastOpt
Schanzenstr. 36
D–20357 Hamburg
Germany
Michael@FastOpt.com

Andrea Walther
Technische Universität Dresden
Fachrichtung Mathematik
Institut für Wissenschaftliches Rechnen
D–01062 Dresden
Germany
awalther@math.tu-dresden.de

Paul J. Werbos
National Science Foundation
4201 Wilson Boulevard
ECS Division, Room 675
Arlington, VA 22203
USA
pwerbos@nsf.gov

Nathan Winslow
Global Modeling and Assimilation Office, Code 610.1
Goddard Space Flight Center
Greenbelt, MD 20771
USA

Jutta Wyes
Process Systems Engineering
RWTH Aachen University
D–52056 Aachen
Germany
wyes@lpt.rwth-aachen.de

Ying Xiao
School of Computer Science
University of Oklahoma
200 Felgar Street
Norman, OK 73019–6151
USA
ying_xiao@ou.edu

Ming Xue
Center for Analysis and Prediction of Storms
University of Oklahoma
Sarkeys Energy Center, Suite 1110
100 East Boyd Street
Norman, OK 73019–1011
USA
mxue@ou.edu

Perspectives on Automatic Differentiation: Past, Present, and Future?

Louis B. Rall

University of Wisconsin – Madison, Madison, WI, USA
rall@math.wisc.edu

Summary. Automatic (or algorithmic) differentiation (AD) is discussed from the standpoint of transformation of algorithms for evaluation of functions into algorithms for evaluation of their derivatives. Such finite numerical algorithms are commonly formulated as computer programs or subroutines, hence the use of the term "automatic." Transformations to evaluate derivatives are thus based on the well-known formulas for derivatives of arithmetic operations and various differentiable intrinsic functions which constitute the basic steps of the algorithm. The chain rule of elementary calculus then guarantees the validity of the process. The chain rule can be applied in various ways to obtain what are called the "forward" and "reverse" modes of automatic differentiation. These modes are described in the context of the early stages of the development of AD, and a brief comparison is given. Following this brief survey, a view of present tasks and future prospects focuses on the need for further education, communication of results, and expansion of areas of application of AD. In addition, some final remarks are made concerning extension of the method of algorithm transformation to problems other than derivative evaluation.

Key words: Numerical algorithms, algorithm transformation, history

The perspectives on automatic differentiation (AD) presented here are from a personal point of view, based on four decades of familiarity with the subject. No claim is made of comprehensive or complete coverage of this now large subject, such a treatment would require a much more extensive work; hence, the question mark in the title. In the time frame considered, AD has gone through the stages listed by Bell [30] in the acceptance of a useful technique: "It is utter nonsense; it is right and can be readily justified; everyone has always known it and it is in fact a trivial commonplace of classical analysis."

It is convenient to adopt as viewpoints on work in AD the classification given by Corliss in the preface to [42]: Methods, Applications, and Tools. Methods relate to the underlying theories and techniques, applications are

what motivate the work, and the final results in terms of computer software are the tools which actually produce solutions to problems.

1 The Algorithmic Approach

Before the middle of the 17th century, algebraic mathematics was based on *algorithms*, that is, step-by-step recipes for the solution of problems. A famous example is Euclid's algorithm for the g.c.d. of two integers, which goes back to the middle of the 3rd century B.C. The term "algorithm" comes from the name of the Islamic mathematician Mohammed ibn Mûsâ al-Khowârizmî, who around 825 A.D. published his book *Hisâb al-jabr w'al-muqâ-balah*, the title of which also gave us the word "algebra."

For centuries, geometers had the advantage of using intuitively evident visual symbols for lines, circles, triangles, etc., and could thus exploit the power of the human brain to process visual information. By contrast, human beings perform sequential processing, such as adding up long columns of numbers, rather slowly and poorly. This made what today are considered rather trivial algebraic operations opaque and difficult to understand when only algorithms were available. This changed in the 17th century with the development and general use of formulas to make algebra visible, and led to the rapid development of mathematics. In particular, I. Newton and G. Leibniz introduced calculus early in this age of formalism. Although their concept of derivative has been put on a sounder logical basis and generalized to broader situations, their basic formulas still underlie AD as practiced today.

The power of the choice of suitable notation and the manipulation of the resulting formulas to obtain answers in terms of formulas was certainly central to modern mathematics over the last 350 years, and will continue to be one of the driving forces of future progress. However, the introduction of the digital computer in the middle of the 20th century has brought the return of the importance of algorithms, now in the form of computer programs. This points to the problem of finding methods for manipulation of algorithms which are as effective as those for formulas.

In modern notation, a *finite* algorithm generates a sequence

$$s = (s_1, s_2, s_3 \ldots, s_n) \ , \tag{1}$$

where s_1 is its *input*, the result s_i of the ith *step* of the algorithm is given by

$$s_i = \phi_i(s_1, \ldots, s_{i-1}), \quad i = 2, \ldots, n \ , \tag{2}$$

where ϕ_i is a function with computable result, and the *output* s_n of the algorithm defines the function f such that $s_n = f(s_1)$. The number n of *steps* of the algorithm may depend on the input s_1, but this dependence will be ignored. For a *finite numerical algorithm* (FNA), the functions ϕ_i belong to a given set Ω of arithmetic operations and certain other computable (intrinsic)

functions. Arithmetic operations are addition, subtraction, multiplication, and division, including the cases of constant (literal) operands. With this in mind, it is sufficient to consider linear combinations, multiplication, and division. For brevity, elements of Ω will be called simply "operations."

This definition of an FNA is easily generalized slightly to the case that the input is a p-vector and the output is a q-vector, or, alternatively, the algorithm s has p inputs $s_1, \ldots, s_p$ and q outputs $s_{n-q+1}, \ldots, s_n$. Such algorithms model computer programs for numerical computations.

In general, it is much easier to grasp the significance of a formula such as

$$f(x, y) = (xy + \sin x + 4)(3y^2 + 6) \ , \tag{3}$$

for a function than a corresponding FNA for its evaluation:

$$\begin{array}{ll} s_1 = x, & s_6 = s_5 + 4, \\ s_2 = y, & s_7 = \mathrm{sqr}(s_2), \\ s_3 = s_1 \times s_2, & s_8 = 3 \times s_7, \\ s_4 = \sin(s_1), & s_9 = s_8 + 6, \\ s_5 = s_3 + s_4, & s_{10} = s_6 \times s_9 \ , \end{array} \tag{4}$$

sometimes called a *code list* for $f(x, y)$. In (4), $\mathrm{sqr}(y) = y2$ has been included as an intrinsic function. However, it is important to note that some functions are defined by computer programs which implement algorithms with thousands or millions of steps and do not correspond to formulas such as (3) in any meaningful way. It follows that algorithms provide a more general definition of functions than formulas.

2 Transformation of Algorithms

In general terms, the transformation of an FNA s into an FNA

$$S = (S_1, S_2, \ldots, S_N)$$

defines a corresponding function F such that $S_N = F(S_1)$. Such a transformation is *direct* if the functions ϕ_i in (1) are replaced by appropriate functions Φ_i on a one-to-one basis. For example, when an algorithm is executed on a computer, it is automatically transformed into the corresponding algorithm for finite precision (f.p.) numbers, which can lead to unexpected results. Other direct transformations from the early days of computing are from single to double or multiple precision f.p. numbers, real to complex, and so on. Another direct transformation is from real to interval, called the *united extension* of the function f by Moore (see [378, Sect. 11.2, pp. 108–113] and [379]): Here, $S_1 = [a, b]$ and $S_N = F(S_1) = [c, d]$, where a, b, c, d are computed f.p. numbers and $f(s_1) \in S_N$ for each $s_1 \in S_1$. The point here is that f.p. numbers can actually be computed which bound the results of exact real algorithms.

The bounds provided by the united extension are guaranteed, but not always useful. Considerable effort has gone into finding interval algorithms S which provide tighter bounds for the results of real algorithms s, but details are far beyond the scope of this paper. These and other more elaborate algorithm transformations make use of replacement of operations by corresponding FNAs if necessary.

From this perspective, AD consists of transformation of algorithms for functions into algorithms for their derivatives. An immediate consequence of the definition of FNAs and the chain rule (which goes back to Newton and Leibniz) is the following

Theorem 1. *If the derivatives s_i' of the steps s_i of the FNA s can be evaluated by FNAs with operations in Ω', then the derivative*

$$s_n' = f'(s_1)s_1'$$

can be evaluated by an FNA with operations in Ω'.

Note: Substitute "formula" for FNA to get symbolic differentiation as taught in school.

As far as terminology is concerned, obtaining an FNA for the derivative is essentially *algorithmic differentiation* [225]. The intent to have a computer do the work of evaluation led to calling this *automatic differentiation* [136, 227, 450], and *computational differentiation* [42] is also perfectly acceptable. In the following, AD refers to whichever designation one prefers.

3 Development of AD

The basic mathematical ideas behind AD have been around for a long time. The methodologies of AD have been discovered independently a number of times by different people at various times and places, so no claim is made here for completeness. Other information can be found in the paper by M. Iri [281] on the history of AD. Although the methodology of AD could well have been used for evaluation of derivatives by hand or with tables and desk calculators, the circuitous method of first deriving formulas for derivatives and then evaluating those seems to have been almost universally employed. Consequently, the discussion here will be confined to the age of the digital computer.

Starting about 1962, the development of AD to the present day can be divided approximately into four decades. In the first, the simple-minded direct approach known as the forward mode was applied to a number of problems, principally at the Mathematics Research Center (MRC) of the University of Wisconsin-Madison as described later in [450]. There followed a slack period marked by lack of acceptance of AD for reasons still not entirely clear. However, interest in AD had definitely revived by 1982 due to improvements in programming techniques and the introduction of the efficient reverse mode.

Much of the progress in this era is due to the work of Andreas Griewank and his colleagues at Argonne National Laboratory (ANL). This was followed by explosive growth of work in techniques, tools, and applications of AD, as recorded in the conference proceedings [42, 136, 227], the book [225] by Griewank, and the present volume. A useful tool in the development of AD following 1980 is the *computational graph*, which is a way of visualizing an algorithm or computer program different from formulas. For example, Fig. 1 shows a computational graph for the algorithm (4). This type of graph is technically known as a directed acyclic graph (DAG), see [225]. Transformations of the algorithm such as differentiation in forward or reverse mode can be expressed as modifications of the computational graph, see for example, [280]. As indicated above, the forward and reverse modes reflect early and later stages in the development of AD, and will be considered below in more detail.

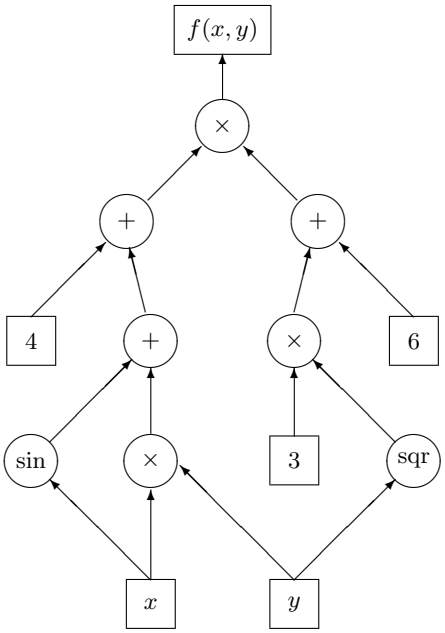


Fig. 1. A Computational Graph for $f(x, y)$.

3.1 The Forward Mode

This mode of AD consists essentially of step-by-step replacement of the operations in the FNA s for the function f by operations or FNAs for their corresponding derivatives. The result will thus be an FNA for derivatives of the result $s_n = f(s_1)$. This follows the order in which the computer program for evaluation of the function f is executed. In fact, before the introduction of

compilers with formula translation, programmers using machine or assembly language had to program evaluation of functions such as (3) in the form of a code list (4). The forward mode of AD reflects this early method of computer programming.

Early workers in AD were R. E. Moore at Lockheed Missiles and Space Company and, later and independently, R. E. Wengert and R. D. Wilkins of the Radio Guidance Operation of the General Electric Company. The work at these two locations had different motivations, but both were carried out in forward mode based on direct conversion of a code list into a sequence of subroutine calls. In reverse historical order, the method of Wengert [530] and the results of Wilkins [555] will be discussed first.

The group at General Electric was interested in perturbations of satellite motion due to nonuniformities in the gravitational field of the earth and checking computer programs which used derivatives obtained by hand. For a function $f(x_1, \ldots, x_d)$, it is often useful to approximate the difference

$$\Delta f = f(x_1 + \Delta x_1, \ldots, x_d + \Delta x_d) - f(x_1, \ldots, x_d) \; , \tag{5}$$

by the *differential*

$$df = \frac{\partial f}{\partial x_1} \Delta x_1 + \cdots + \frac{\partial f}{\partial x_d} \Delta x_d \; , \tag{6}$$

a linearization of (5) which is accurate for sufficiently small values of the increments $\Delta x_1, \ldots, \Delta x_d$. (It is customary to write dx_j instead of Δx_j in (6) to make the formula look pretty.) Leaving aside the situation that one or more of the increments may not be sufficiently small enough to make (6) as accurate as desired, the values of the partial derivatives $\partial f / \partial x_j$ give an idea of how much a change in the jth variable will perturb the value of the function f, and in which direction. Consequently, the values of these partial derivatives are known as "sensitivities."

Wengert's method used ordered pairs and does not calculate partial derivatives directly. Rather, after initialization of the values (x_j, x_j') of the independent variables and their derivatives, the result obtained is the pair (f, f'), where f is the function value and f' the *total* (or *directional*) derivative

$$f' = \frac{\partial f}{\partial x_1} x_1' + \cdots + \frac{\partial f}{\partial x_d} x_d' \; . \tag{7}$$

Values of individual partial derivatives $\partial f / \partial x_k$ are thus obtained by the initialization (x_j, δ_{jk}), δ_{jk} being the Kronecker delta. Wengert notes that higher partial derivatives can be obtained by applying the product rule to (7) and repeated evaluations with suitable initializations of (x_j, x_j'), (x_j', x_j''), and so on to obtain systems of linear equations which can be solved for the required derivatives.

As an example of the line-by-line programming required (called the "key to the method" by Wengert), starting with $\mathtt{S1}(1) = x$, $\mathtt{S1}(2) = 1$, $\mathtt{S2}(1) =$

y, S2(2) = 0, the computation of (f,$\partial f/\partial x$) of the function (3) would be programmed as

```
CALL PROD(S1, S2, S3)
CALL SINE(S1, S4)
...  ...  ...  ...                    (8)
CALL ADD(S8, 6, S9)
CALL PROD(S6, S9, S10)
```

following the code list (4), and then repeated switching the initializations to $x' = 0$ and $y' = 1$ to obtain $(f, \partial f/\partial y)$. The example given by Wilkins [555] is a function for which 21 partial derivatives are desired. The computation, after modification to avoid overflow, is repeated 21 times, and Wilkins notes the function value is evaluated 20 more times than necessary. The overflow was due to the use of the textbook formula for the derivative of the quotient by Wengert [530]. Wilkins notes an improvement suggested by his coworker K. O. Johnson to differentiate $u/v = uv^{-1}$ as a product was helpful with the overflow problem, and finally Wengert suggested the efficient expression $(u/v)' = (u' - (u/v)v')/v$ which uses the previously evaluated quotient. Also, since the function considered also depends on the time t and contains derivatives w.r.t. t, it is not clear which derivative was calculated, the ordinary total derivative (7) or the ordered derivative

$$\frac{\partial^+ f}{\partial t} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \cdots + \frac{\partial f}{\partial x_d}\frac{\partial x_d}{\partial t} \ ,$$

denoted by $Df/\partial t$ in [483].

Although inefficient, the program using Wengert's method, once corrected, showed there were errors in the program using derivatives obtained by hand. When the latter was corrected, both took about the same computer time to obtains answers which agreed. That Wilkins had to battle f.p. arithmetic shows that "automatic" in the sense of "plug-and-play" does not always hold for AD, as is also well-known in the case of interval arithmetic. Wilkins predicted a bright future for AD as a debugging tool and a stand-alone computational technique. However, AD seems to have hit a dead end at General Electric; nothing more appeared from this group as far as is known. As a matter of fact, the efforts of Wengert and Wilkins had no influence on the subsequent work using AD done at MRC off and on over the next ten years.

Prior to Wengert and Wilkins, R. E. Moore worked on the initial-value problem

$$\dot{x} = f(x,t), \quad x(t_0) = x_0 \ , \tag{9}$$

the goal being to compute f.p. vectors $a(t)$ and $b(t)$ such that the bounds $a(t) \leqq x(t) \leqq b(t)$ are guaranteed. Moore used recurrence relations for the Taylor series expansion of $f(x,t)$ to obtain the Taylor expansion

$$x(t_0 + \tau) = \sum_{k=0}^{m} x_k + R_m \ , \tag{10}$$

of the solution, where

$$x_k = \frac{1}{k!}\frac{d^k x(t_0)}{dt^k}\tau^k \tag{11}$$

is the kth normalized Taylor coefficient of the function $x(t)$, and the remainder term R_m is given by

$$R_m = \frac{1}{(m+1)!}\frac{d^{m+1}x(\vartheta)}{dt^{m+1}}\tau^{m+1}, \quad t_0 \leqq \vartheta \leqq t_0 + \tau\ . \tag{12}$$

Moore used interval arithmetic to bound the round-off error in the computation of the Taylor coefficients (11) and the truncation error (12) on the interval $[t_0, t_0 + \tau]$. In this way, valid assertions could be made about the results of an algorithm carried out in f.p. interval arithmetic. As in the later work of Wengert, the expansion (10) was based on representation of the function $f(x,t)$ by a code list and was programmed as a sequence of subroutine calls such as (8).

Moore presented his results to conferences on error in digital computation held at MRC in 1964 and 1965, see [376,377]. It was recognized immediately that Moore's method also applied to the direct evaluation of partial derivatives of functions of several variables, rather than via total derivatives as done by Wengert. The motivation was automation of Newton's method in d dimensions for approximate solution of $F(x) = 0$ by solving the sequence of linear equations

$$F'(x)(x_{m+1} - x_m) = -F(x_m), \quad m = 0, 1, \dots\ , \tag{13}$$

where the coefficient matrix is the Jacobian $F'(x) = (\partial F_i/\partial x_j)$ of the system of functions $F_i(x),\ i = 1, \dots, d$. The rows of the matrix $F'(x)$ are the gradients $\nabla F_i(x)$ of the corresponding functions $F_i(x)$. Furthermore, in order to apply the theorem of Kantorovich (see [449]) on the convergence of Newton's method, a Lipschitz constant for $F'(x)$ is required. This can be obtained from an upper bound for the ∞-norm of the Hessian operator

$$K \geqq \|F''(x)\| = \left\| \frac{\partial^2 F_i}{\partial x_j \partial x_k} \right\|\ .$$

The necessary bounds were computed using interval arithmetic, so that valid assertions regarding the existence of a solution and a region containing it were obtained as well as the convergence of the Newton sequence (13) when successful.

This program and others written at MRC by an outstanding programming staff supervised by L. Rall incorporated a number of advances over previous efforts in several respects. First of all, the programs accepted expressions (functions) as input and produced the corresponding sequences of subroutine calls internally, thus relieving the user of this unnecessary task. Secondly, gradients and Hessians were vectorized, so only one pass was required to obtain

the value of a function, its gradient vector, and Hessian matrix. First and second derivatives of operations and intrinsic functions were coded explicitly, rather than using Taylor coefficients or the product rule and linear equations as indicated by Wengert. Moore's program for initial-value problems was also modified to accept expressions as inputs. Finally, a program for numerical integration with guaranteed error bounds was written to accept subroutines (which could be single expressions) as input. For more details on the programs written at MRC, see [450].

Also at the University of Wisconsin, G. Kedem wrote his 1974 Ph.D. thesis on automatic differentiation of programs in forward mode, supervised by C. de Boor. It was published in 1980 [302]. For various reasons, work on AD at MRC came to a pause in 1974, and was not taken up again until 1981. This followed lectures given at the University of Copenhagen [450] and a visit to the University of Karlsruhe to learn about the computer language Pascal-SC, developed by U. Kulisch and his group (see [66] for a complete description). This extension of the computer language Pascal permits operator overloading and functions with arbitrary result types, and thus presents a natural way to program AD in forward mode, see [451] for example. Much of this work was done in collaboration with G. Corliss of Marquette University [133]. Another result was an adaptive version of the self-validating numerical integration program written earlier at MRC in nonadaptive form [137]. Funding of MRC was discontinued in 1985, which brought an end to this era of AD.

Another result of the technique of operator overloading was the concept of *differentiation arithmetic*, introduced in an elementary paper by Rall [452]. This formulation was based on operations on ordered pairs (a, a') (as in [530]). In algebraic terms, this showed that AD could be considered to be a derivation of a commutative ring, the rule for multiplication being the product rule for derivatives. Furthermore, M. Berz noticed that in the definition $(a, a') = a(1, 0) + a'(0, 1)$, the quantity $(1, 0)$ is a basis for the real numbers and, in the lexicographical ordering, $(0, 1)$ is a nonzero quantity less than any positive number and hence satisfies the classical definition of an infinitesimal. Starting from this observation, Berz was able to frame AD in terms of operations in a Levi-Civita field [37].

3.2 The Reverse Mode

Along with the revival of the forward mode of AD after 1980, the reverse mode came into prominence. As in the case of the forward mode, the history of the reverse mode is somewhat murky, featured by anticipations, publication in obscure sources [420], Ph.D. theses which were unpublished [487] or not published until much later. For example, the thesis of P. Werbos [532] was not published until twenty years later [543]. Fortunately, the thesis of B. Speelpenning [487] attracted the attention of A. Griewank at ANL, and further notice was brought to the reverse mode by the paper of M. Iri [280].

The basic idea of the reverse mode for the case the algorithm (1) has d inputs and one output is to apply the chain rule to calculate the "adjoints,"

$$\frac{\partial s_n}{\partial s_n}, \frac{\partial s_n}{\partial s_{n-1}}, \ldots, \frac{\partial s_n}{\partial s_d}, \ldots, \frac{\partial s_n}{\partial s_1} \ ,$$

which provide in reverse order the components of the gradient vector

$$\nabla f = \nabla s_n = \left(\frac{\partial s_n}{\partial s_1}, \ldots, \frac{\partial s_n}{\partial s_d} \right) \ .$$

The reverse mode resembles symbolic differentiation in the sense that one starts with the final result in the form of a formula for the function and then applies the rules for differentiation until the independent variables are reached. For example, (3) is a product, so the factors

$$\frac{\partial s_{10}}{\partial s_9} = s_6 = xy + \sin x + 4, \quad \frac{\partial s_{10}}{\partial s_6} = s_9 = 3y2 + 6 \ ,$$

are taken as new differentiation problems, with the final results of each composed by the product rule to obtain

$$\begin{aligned} \frac{\partial f}{\partial x} &= \frac{\partial s_{10}}{\partial s_1} = (y + \cos x)(3y2 + 6), \\ \frac{\partial f}{\partial y} &= \frac{\partial s_{10}}{\partial s_2} = x(3y2 + 6) + 6y(xy + \sin x + 4) \ , \end{aligned} \tag{14}$$

which can be "simplified" further if desired. Applied to the example code list (4), the reverse form yields

$$\begin{aligned} \frac{\partial s_{10}}{\partial s_{10}} &= 1, & \frac{\partial s_{10}}{\partial s_6} &= s_9, \\ \frac{\partial s_{10}}{\partial s_9} &= s_6, & \frac{\partial s_{10}}{\partial s_5} &= \frac{\partial s_{10}}{\partial s_6}\frac{\partial s_6}{\partial s_5} = s_9 \times 1, \\ \frac{\partial s_{10}}{\partial s_8} &= \frac{\partial s_{10}}{\partial s_9}\frac{\partial s_9}{\partial s_8} = s_6 \times 1, & \frac{\partial s_{10}}{\partial s_4} &= \frac{\partial s_{10}}{\partial s_5}\frac{\partial s_5}{\partial s_4} = s_9 \times 1, \\ \frac{\partial s_{10}}{\partial s_7} &= \frac{\partial s_{10}}{\partial s_8}\frac{\partial s_8}{\partial s_7} = s_6 \times 3, & \frac{\partial s_{10}}{\partial s_3} &= \frac{\partial s_{10}}{\partial s_5}\frac{\partial s_5}{\partial s_3} = s_9 \times 1 \ , \end{aligned}$$

with the final results

$$\begin{aligned} \frac{\partial s_{10}}{\partial s_2} &= \frac{\partial s_{10}}{\partial s_7}\frac{\partial s_7}{\partial s_2} + \frac{\partial s_{10}}{\partial s_3}\frac{\partial s_3}{\partial s_2} = (3s_6)(2s_2) + s_9 s_1, \\ \frac{\partial s_{10}}{\partial s_1} &= \frac{\partial s_{10}}{\partial s_4}\frac{\partial s_4}{\partial s_1} + \frac{\partial s_{10}}{\partial s_3}\frac{\partial s_3}{\partial s_1} = s_9 s_1 + s_9 s_2 \ , \end{aligned}$$

the same values as given by (14). Even in this simple case, fewer operations are required by this computation than forward evaluation of the algorithm (4) using the pairs $S_i = (s_i, \nabla s_i)$. Programming of the reverse mode is more elaborate than the forward mode, however, operator overloading can be done in reverse mode as in ADOL-C [288].

3.3 A Comparison

Both forward and reverse modes have their places in the repertoire of computational differentiation, and are sometimes used in combination. The efficiency of the reverse mode is sometimes offset by the necessity to store results of a long algorithm, see [429,550], for example. A theoretical comparison has been given by Rall [174, pp. 233–240] based on the matrix equivalent of the computational graph. If the algorithm (1) is differentiable, then its Jacobian matrix $J = (\partial s_i/\partial s_j)$ is of the form $J = I - K$, where K is lower-triangular and sparse. The eigenvalues of J are all equal to 1, and $K^{\nu+1} = 0$ for some index ν. The row vector

$$R = [0 \cdots 0 \ \nabla s_n]$$

is a left eigenvector of J, and the columns of the $n \times d$ matrix C with rows $\nabla s_1, \nabla s_2, \ldots, \nabla s_n$ are right eigenvectors of J. The reverse and forward modes consist of calculating these eigenvectors by the power method. The reverse mode starts with $R_0 = [0 \ \cdots \ 0 \ 1]$ and proceeds by $R_k = R_{k-1}J$ and terminates with $R_\mu = R$. Similarly, the forward mode starts with $C_0 = [\nabla s_1^T \ \cdots \ \nabla s_d^T \ 0 \ \cdots \ 0]^T$, proceeds by $C_k = JC_{k-1}$, and terminates with $C_\mu = C$. The difference in computational effort is immediately evident.

4 Present Tasks and Future Prospects

The future of AD depends on what is done now as well as what transpired in the past. Current tasks can be divided into four general categories: Techniques, education, communication, and applications. Some brief remarks will be devoted to each of these topics.

4.1 Techniques

The basic techniques of AD are well understood, but little attention has been devoted to accuracy, the assumption being that derivatives are obtained about as accurately as function values. The emphasis has been on speed and conservation of storage. Increasing speed by reducing the number of operations required is of course helpful, since the number of roundings is also decreased. Advantage can also be taken of the fact that $ab + c$ is often computed with a single rounding. Even more significant would be the provision of a long accumulator to evaluate the dot product

$$u \cdot v = \sum_{i=1}^{d} u_i v_i$$

of d-vectors u and v with a single rounding as implemented originally in Pascal-SC [66]. This enables many algebraic operations including solution of

linear systems to be carried out with high accuracy. For example, the components of the gradient $\nabla(u \cdot v)$ of a dot product can be expressed as dot products and thus computed with a single rounding. Also, if $x = L^{-1}y$ is the solution of a nonsingular system of equations $Lx = y$, then its gradient ∇x is given by the generalization of the division formula

$$\nabla x = L^{-1}\nabla y - L^{-1}(\nabla L)L^{-1}y = L^{-1}(\nabla y - (\nabla L)x) \ ,$$

(see [449]), where ∇L is a $d \times d$ matrix of gradients and ∇y is a d-vector of gradients. Of course, it is unnecessary to invert L, the system of equations $L\nabla x = \nabla y - (\nabla L)x$ can be solved by the same method as for $Lx = y$.

4.2 Education

It was discouraging throughout the 1970's that the work done on AD by Moore, Wengert, and the then state of the art programs written by the MRC programming staff were ignored and even disparaged. Presentations at conferences were met with disinterest or disbelief. One reason advanced for this was the wide-spread conviction that if a function was represented by a formula, then a formula for its derivative was necessary before its derivative could be evaluated. Furthermore, the differentiation of a function defined only by an algorithm and not by a formula seemed beyond comprehension. A few simple examples could be incorporated into elementary calculus courses to combat these fallacies. As mentioned above, the standard method taught for differentiation of functions defined by formulas essentially proceeds in reverse mode. The forward mode uses the way the final result $f(x)$ is computed from the given value of x and shows that $f'(x)$ can be evaluated in the same step-by-step fashion. Furthermore, given the definitions (2), the values $x = s_1, s_2, \ldots, s_n = f(x)$ of the steps in the evaluation of $f(x)$ can be used in the reverse mode to obtain the same value of $f'(x)$. Then, for example, Newton's method can be introduced as an application of use of derivative values without the necessity to obtain formulas for derivatives. All of this can be done once the basic formulas for differentiation of arithmetic operations and some elementary functions have been taught.

It is easy to prepare a teaching "module" for AD on an elementary level. The problem is to have it adopted as part of an increasingly crowded curriculum in beginning calculus. This means that teachers and writers of textbooks on calculus have to first grasp the idea and then realize it is significant. Thus, practitioners of AD will have to reach out to educators in a meaningful way. Otherwise, there will continue to be a refractory "formulas only" community in the computational sciences who could well benefit from AD.

Opportunities to introduce AD occur in other courses, such as differential equations, optimization, and numerical analysis. Reverse mode differentiation is a suitable topic for programming courses, perhaps on the intermediate level. An informal survey of numerical analysis and other textbooks reveals that

most recommend *against* the use of derivatives, in particular regarding Newton's method and Taylor series solution of differential equations. The reason advanced is the complexity of obtaining the "required" formulas for derivatives and Taylor coefficients by hand. An exception is the recent textbook on numerical analysis by A. Neumaier [411], which begins with a discussion of function evaluation and automatic differentiation. A definite opportunity exists to introduce AD at various levels in the curricula of various fields, including business, social and biological sciences as well as the traditional physical sciences and engineering fields. This is particularly true now that most instruction is backed up by software pertinent to the subject.

4.3 Communication and Applications

An additional reason for the slow acceptance of AD in its early years was the lack of publication of results after the papers of Wengert and Wilkins [530,555]. For example, the more advanced programs written at MRC were described only in technical reports and presented at a few conferences sponsored by the U. S. Army, but not widely disseminated. The attitude of journal editors at the time seemed to be that AD was either "a trivial commonplace of classical analysis," or the subject was completely subsumed in the paper by Wengert [530]. In addition, the emphasis on interval arithmetic and assertions of validity in the MRC approach had little impact on the general computing community, which was more interested in speed than guarantees of accuracy. Furthermore, the MRC programs were tied rather closely to the computer available at the time, standards for computer and interval arithmetic had not yet been developed. The uses of AD for Taylor series in Moore's 1966 book [378] and Newton's method in Rall's 1969 book [449] were widely ignored.

A striking example of lack of communication was shown in the survey paper by Barton, Willers, and Zahar, published in 1971 [461, pp. 369–390]. This valuable and interesting work on Taylor series methods traced the use of recurrence relations as employed by Moore back to at least 1932 and included the statement, "... adequate software in the form of automatic programs for the method has been nonexistent." The authors and the editor of [461] were obviously unaware that Moore had such software running about ten years earlier [375], and his program was modified by Judy Braun at MRC in 1968 to accept expressions as input, which made it even more automatic.

Another impediment to the ready acceptance of Moore's interval method for differential equations was the "wrapping effect" [377]. This refers to unreasonably rapid increase in the width of the interval $[a(t), b(t)]$ to make these bounds for the solution useless. Later work by Lohner [343] and in particular the Taylor model concept of Berz and Makino [266,346,347] have ameliorated this situation to a great extent.

Fortunately, publication of the books [225, 450], and the conference proceedings [42, 136, 227], and the present volume have brought AD to a much

wider audience and increased its use worldwide. The field received an important boost when the precompiler ADIFOR 2.0 by C. Bischof and A. Carle was awarded the J. H. Wilkinson prize for mathematical software in 1995 (see SIAM News, Vol. 28, No. 7, August/September 1995). The increasing number of publications in the literature of various fields of applications is likewise very important, since these bring AD to the attention of potential users instead of only practitioners. These books and articles as cited in their extensive bibliographies show a large and increasing sphere of applications of AD.

5 Beyond AD

Perhaps the bright future for AD predicted 40 years ago by Wilkins has arrived or is on the near horizon. There is general acceptance of AD by the optimization and interval computation communities. With more effort directed toward education, the use of AD will probably become routine. Perhaps future generations of compilers will offer differentiation as an option, see [400]. Directions for further study are to use the lessons learned from AD to develop other algorithm transformations. A step in this direction by T. Reps and L. Rall [459] is the algorithmic evaluation of divided differences

$$[x, h]f = \frac{f(x+h) - f(x)}{h} \ . \tag{15}$$

Direct evaluation of (15) in f.p. arithmetic is problematical, whereas algorithmic evaluation is stable over a wide range of values, and approaches the value of the AD derivative $f'(x)$ as $h \to 0$. In fact, for $h = 0$, the divided difference formulas reduce to the corresponding formulas for derivatives. In the use of divided differences to approximate derivatives, (15) is inaccurate due to truncation error for h large, and due to roundoff error for h small. On the other hand, the use of differentials (6) obtained by AD to approximate differences (5) has the same problems. Thus, it is useful to have a method to compute differences which does not suffer loss of significant digits by cancelation to the extent encountered in direct evaluation.

Other goals for algorithm transformation are suggested by the "ultra-arithmetic" proposed by W. Miranker and others [295]. Algorithms for functions represented by Fourier-type series can be used to obtain the coefficients of the series expansions, much like what has already been done for Taylor series. In other words, the transformation of an FNA can be accomplished once the appropriate transformations of arithmetic operations and intrinsic functions involved are known. As initially realized by Wengert [530], this is indeed the key to the method.