AUTOMATIC DIFFERENTIATION OF ALGORITHMS:

THEORY, IMPLEMENTATION, AND APPLICATION

Edited by Andreas Griewank
Argonne National Laboratory

George F. Corliss
Marquette University

siam. Philadelphia

Society for Industrial and Applied Mathematics

AUTOMATIC DIFFERENTIATION

OF ALGORITHMS:

THEORY, IMPLEMENTATION, AND APPLICATION

Proceedings of the first SIAM Workshop on Automatic Differentiation, held in Breckenridge, Colorado, January 6–8, 1991.

This conference was supported in part by the Department of the Air Force under grant AFOSR-91-0064 and the U. S. Army Reaseach Office under grant DAAL-03-90-G-0224.

Library of Congress Cataloging-in-Publication Data

Automatic differentiation of algorithms: theory, implementation, and application / edited by Andreas Griewank and George F. Corliss.
p. cm.
"Proceedings of the SIAM Workshop on the Automatic Differentation of Algorithms, held January 6–8, 1991, in Breckenridge, Colorado"--T.p. verso.
Includes bibliographical references.
ISBN 0-89871-284-X
1. Calculus, Differential–Data processing–Congresses. I. Griewank, Andreas. II. Corliss, George F. III. Society for Industrial and Applied Mathematics. IV. SIAM Workshop on the Automatic Differentation of Algorithms (1991 Breckenridge, Colo.)
QA304.A8 1992
515'.33'0285–dc20 91-43140

A System for the Differentiation of Fortran Code and an Application to Parameter Estimation in Forest Growth Models

Oscar García*

Abstract. An automatic differentiation system, GRAD, is described. Given a Fortran subprogram for computing a function, it generates a subprogram that computes partial derivatives. The APL computer language was used in the implementation.

The performance of automatic differentiation in fitting growth models for intensively managed forest plantations is examined. The models consist of a system of stochastic differential equations, and parameters are estimated by maximum-likelihood using a general-purpose variable-metric optimization procedure. Compared to central difference approximations, the use of derivatives generated by GRAD in the optimization reduced computing time by a factor of 4 on an 80386/80387 microcomputer and by a factor of 6 on a MicroVAX 3500. GRAD was found superior to JAKEF in this type of problem.

GRAD combines the forward mode of automatic differentiation with symbolic manipulation. A conceptual framework capable of describing these hybrid strategies is presented, and their advantages are discussed.

Keywords: Automatic differentiation, symbolic differentiation, computer algebra, statistics, estimation, optimization, forestry, maximum-likelihood, stochastic differential equations, APL, GRAD, JAKEF.

1 Introduction. In an effort to speed-up the estimation of parameters in complex forest growth models, an automatic differentiation system, GRAD, was developed [Garc89a]. It takes as input a Fortran program unit (usually a subroutine or function subprogram) that computes the value of a function, and produces as output another Fortran program unit that computes partial derivatives with respect to specified independent variables.

The paper is in three parts. First, the methods used in GRAD and its implementation are described, and examples of output are shown. In the second part I discuss the application to growth models, and compare the performance of the generated differentiation code with the use of finite difference approximations. In this type of problem GRAD is found to be superior to JAKEF [Hill85a], one of the better-known automatic differentiation systems. Finally, a framework for the description and analysis of automatic differentiation strategies

*Forest Research Institute, Rotorua, New Zealand.

is presented. A state-space interpretation is used, instead of the more common treatment through computational graphs. This allows for the use of symbolic manipulation at the statement level, as implemented in GRAD.

2 The GRAD automatic differentiator. The basic idea was inspired by Wengert [Weng64a]. He suggested substituting for each operator a routine, that in addition to computing the result of the operation would also produce the values of the derivatives. A related approach was implemented by hand in order to compute first and second partial derivatives in a parameter estimation program ([Garc83a] and Appendix 2). The problem required the computation of the derivatives of a function defined by a complicated Fortran subroutine (the log-likelihood function) with respect to a number of independent variables (the model parameters). A subroutine to compute the derivatives was built by following each assignment statement in the function evaluation routine by statements computing the derivatives of the left-hand-side variable (usually an intermediate variable). For example.

```
      D = A * B + SIN(C)
```

would be followed by

```
      D1 = A1 * B + A * B1 + COS(C) * C1
      D2 = A2 * B + A * B2 + COS(C) * C2
      etc.,
```

where the Ak, Bk, and Ck are partial derivatives with respect to the k-th independent variable. computed in previous statements (with obvious simplifications where the partials for some terms do not exist). The function derivatives sought follow the final function value assignment, at the end of the subroutine. Unlike Wengert's, this method produces stand-alone code, without the overheads of calling routines in a run-time package.

GRAD essentially automates this procedure. The statements that compute derivatives precede instead of follow each assignment from the input, in order to handle correctly statements that re-define variables, as in

```
      A = A * B .
```

For each assignment statement in the input, the Fortran code to compute the relevant derivatives is generated by symbolic manipulation of the right-hand-side. The symbolic differentiation algorithm is based on recursive descent parsing [Davi81a, for example]. It does not need to be as sophisticated as those in computer algebra systems [Char91a,Gold91a], since maximum simplification of expressions is not required; most redundancies will be removed by an optimizing Fortran compiler. Unnecessary parentheses, however, are avoided as much as possible because parentheses in Fortran fix the order of operations, inhibiting optimization. Note also that a good optimizing compiler will take care of common subexpressions across statements, such as COS(C) in the example above.

Identifiers for the derivatives are formed by appending to the variable name a user-defined delimiter character and the number of the independent variable (see the examples in Appendix 1, where the delimiter is _). Identifiers exceeding the 6-character Fortran standard maximum length might need to be changed by the user (a warning is produced), although many compilers can accept long variable names.

A list of left-hand-side variables encountered and derivatives generated is kept in an internal data structure as the statements are processed. In order to minimize computing and storage requirements, new statements are initially generated only for non-zero derivatives. In some instances this would not produce correct results, and multiple passes may be needed to initialize at zero some of the derivatives. For example, with one pass the following statements would generate incorrect code (X is the first independent variable and this is the first appearance of A; derivatives of VAR with respect to the k-th independent variable are identified as VAR_k):

```
      IF (I .LT. 5) GO TO 10          IF (I .LT. 5) GO TO 10
        A = 0.                          A = 0.
        GO TO 20                        GO TO 20
```

```
10 CONTINUE                  10 CONTINUE
     A = X          ==>           A_1 = 1.
20 CONTINUE                       A = X
     B = A + 1               20 CONTINUE
                                  B_1 = A_1
                                  B = A + 1
```

Here `A_1 = 0.0` would be needed following the `IF` statement. Similar problems may arise from loops, and in other situations as in Problem E in Appendix 1. GRAD handles these cases automatically with multiple passes over the input code. The need for another pass is detected when a derivative is encountered for the first time for a variable that had been used on the left-hand-side of a previous assignment. Keeping the list of derivatives from the previous pass causes that new derivative to be initialized at zero in the appropriate place.

A result of the myopic strategy of examining only assignment statements, one at a time, was a relatively simple and efficient system. However, a few restrictions must be observed in the source coding:

- User defined functions involving the independent variables are not supported.
- The independent variables must be scalars or array elements with constant subscripts.
- Assignments involving independent variables directly or indirectly are not allowed in IF or other control statements.
- Labels are not allowed in assignment statements (use `CONTINUE`).

In addition, the header of the output routine must be edited manually to change its name and to include the derivatives as arguments. It may also be necessary to declare the new variables if default types or `IMPLICIT` declarations are not used. Most of these limitations could be eliminated by adding appropriate pre- and/or post-processing steps to the current implementation.

If the input subroutine computes a vector-valued function, GRAD produces the elements of the Jacobian matrix. Second or higher derivatives can be generated simply by running the output through GRAD again. The resulting code, however, will perform redundant computations, since (for second derivatives) two versions of the gradient and of the off-diagonal Hessian elements are generated. It would not be difficult to eliminate this redundancy in a post-processing step, but this has not been implemented yet.

GRAD was written in APL, using STSC's APL*PLUS interpreter for IBM PC or compatible microcomputers [STSC89a]. This is transparent to the user, however, and no knowledge of APL is required to use it. The high level and power of the APL computer language made it possible to complete the implementation in a very short time. The system also runs with the inexpensive Pocket APL [Turn85a] and other STSC APL interpreters for various computers, and a version for the free I-APL interpreter is available. Both versions of GRAD can be obtained from the author, free.

3 Application and performance.

3.1 The problem. The development of GRAD was motivated by the need to speed-up the estimation of parameters in a series of growth models for forest plantations. The models are systems of stochastic differential equations, with parameters estimated by maximum-likelihood. Given a set of data, parameter estimation is an unconstrained function optimization problem, finding the parameter values that minimize the negative log-likelihood. The models and estimation procedures are described in more detail in Appendix 2 and in the references.

For each data set, there are typically many parameter estimation runs, with variations of the basic model involving different numbers of state variables, the fixing at zero of various subsets of the parameters, and other model changes. It is also advisable to repeat the procedure with different starting points to guard against the possibility of local optima. The number of parameters vary between 9 and 20.

The optimization is carried out with a general-purpose numerical optimization routine. After some unsuccessful experiences with a finite differences implementation of Fletcher and Powell's algorithm [Lill70a], and with Nelder and Mead's simplex procedure ([ONei71a], see [Garc79a]), Hatfield Polytechnic's OPVM routine has performed well [Bigg71a,Bigg73a, NOC76a]. OPVM is a Fortran subroutine that uses a quasi-Newton or variable-metric unconstrained optimization method. For the gradient it can use analytic first derivatives, or approximate them with central differences in the auxiliary subroutine OPND1. Other more recent optimization routines have not been tested because they lack an essential feature of OPVM: if the function evaluation routine cannot compute the value at a given point, it can set a flag, and OPVM then reduces the step length and tries again. This is necessary because often trial points cause floating point exceptions, typically from out of range arguments in exponentials and other functions. In addition, some parameter values result in complex eigenvalues for the **A** matrix (Appendix 2), unacceptable on physical grounds. The alternative of supplying step bounds is generally unsatisfactory.

The likelihood function to be optimized in these models is fairly complex. As much of the computation as possible is done in a pre-processing step, storing transformed data in an array. Still, the function evaluation subroutines called by the optimization procedure contain some 130 to 180 Fortran statements. This size, together with the frequent modifications to the programs, made impractical the hand-coding of analytic derivatives. Therefore, the OPND1 difference approximations were used.

Each function evaluation includes a loop over hundreds, or even thousands of observations. An approximation of the gradient by central differences requires a number of function evaluations equal to twice the number of variables (parameters). With the use of more complex model forms and larger data sets, the necessity of many lengthy over-night computer runs increased costs and slowed down progress considerably. In 1988, GRAD was developed in an attempt to improve computing turn-around.

3.2 Tests and results. The use of analytic derivatives generated by GRAD and of difference approximations has been compared on three parameter estimation problems (Table 1). Problem A is one of the simpler models, with a moderate amount of data [Garc84a]. Problem B is a more complex model [Garc89a], typical of those that motivated the automatic differentiation approach. The runs with A and B started from reasonable estimates, with those for A derived from the solution of B, and vice-versa. Problem C tested a more general form for the stochastic structure, with additional parameters to be estimated [Garc79a,Garc84a]. The starting point for C was the parameters estimated with the simpler form, and the optimization procedure stopped with only small changes in the log-likelihood and failing the convergence test, showing the problem to be ill-conditioned (over-parametrized).

The computations were performed in double precision on an AT-compatible 20 MHz 80386 microcomputer with an 80387 coprocessor and Microsoft Fortran 4.10. Problem B was also run on a MicroVAX 3500 with VAX Fortran. The problems are reasonably well scaled, with variables and objective value not very far from one. The step size for the finite differences was set to 10^{-2} for runs A and B, and to 10^{-4} for C. Previous experience had shown that the step size is not critical, and that these values are satisfactory.

The third line in Table 1 shows the time taken by the automatic differentiation procedure. This does not include the minor manual editing required to change the subroutine header and assign the gradient. The finite differences run on problem B stopped without reducing all gradient components below the value of 10^{-3} specified in the convergence cri-

Problem	A	B	C	B, VAX
Variables (parameters)	9	16	18	16
Observations	339	2093	1655	2093
Gradient subroutine generation (min)	6	10	14	—
Difference approximations				
Function calls	70	301	147	348
Gradient calls	45	184	32	210
Time (minutes)	10.8	890.7	89.9	614.8
Analytic derivatives				
Function calls	72	352	145	384
Gradient calls	45	211	33	221
Time (minutes)	2.7	228.5	21.8	99.3
Difference approximations run-time / analytic derivatives run-time	4.0	3.9	4.1	6.2

Table 1: Parameter estimation runs.

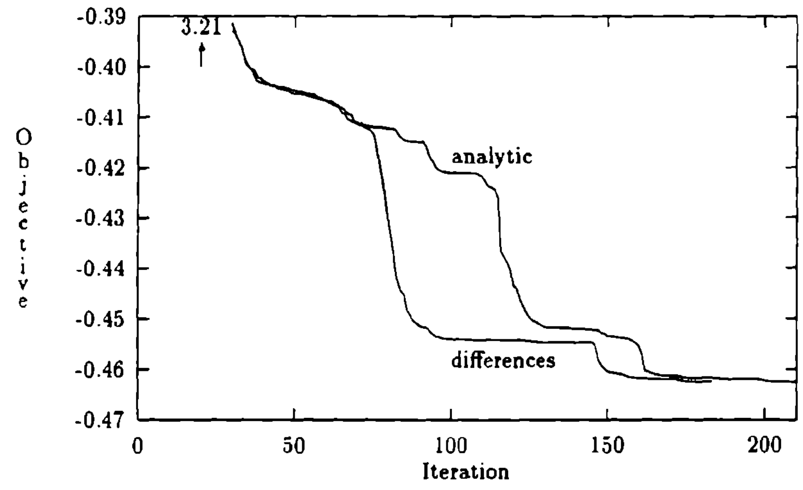

Figure 1: Optimization progress for problem B.

terion, although it was very close. The speed-up factors from using analytic derivatives are given in the last row of Table 1. Some additional improvement could be obtained by modifying the optimization routine to make use of the function values computed by the analytic gradients subroutine.

There was no evidence of improved reliability from using analytic derivatives, except perhaps for the difference in satisfying the stringent convergence test just mentioned. The optimization paths in problems A and C were very similar, but in problem B the difference approximations achieved larger reductions of the objective in fewer iterations (Figure 1).

Table 2 shows what Griewank ([Grie89a]) calls the work ratio, the ratio of the time required to compute the (analytic) gradient to the time required to compute one function evaluation (actually, a function value is also produced together with the gradient). This can be compared to ratios of $n + 1$ and $2n + 1$ for one-sided and central finite differences, respectively, where n is the number of variables. Problem D is a model similar to A, but with more free parameters, and with the data set of C. The computations for D were done on a 12 MHz AT-compatible 80286 microcomputer, with the 80287 coprocessor circuitry

Problem	Variables	Observations	Work ratio
A	9	339	3.0
B	16	2093	5.6
B, VAX	16	2093	3.7
C	18	1655	6.4
D	18	1655	5.2

Table 2: Ratio of gradient to function evaluation times (work ratio).

Problem	Variables	GRAD	JAKEF
D, 100 obs.	18	5.2	8.0
E	4	2.7	11.1
F	3	3.1	11.2

Table 3: Work ratios for GRAD and JAKEF

modified to run at 12 MHz. and Microsoft Fortran 4.10.

Another automatic differentiation system, JAKEF [Hill85a,Jued91a], was also tried. JAKEF uses a reverse differentiation approach [Grie89a], with storage requirements increasing with the number of statements executed in the function evaluation. This number is large here, due to the loop over the observations. In the 640 Kbytes of memory available in a microcomputer under DOS, JAKEF was unable to handle problems of practical size (200 or more observations). In addition, a test with problem D on a subset of 100 observations showed that the code generated by JAKEF was substantially slower (Table 3). The results for two small examples (Appendix 1) are also included in the Table as E and F.

3.3 Discussion. I have not tested alternatives to the direct maximization of the likelihood with a variable-metric algorithm. An attractive possibility are the extensions of Fisher's method of scoring [Zack71a] discussed by Bard [Bard74a] under the name of the Gauss Method. These provide an explicit approximation to the Hessian that can be used in the likelihood optimization. Although more difficult to implement, these techniques might be more efficient. The relative performance of analytic derivatives vs difference approximations, however, is likely to be similar. The same is true for alternative estimation criteria, for example, the maximization of the product of the likelihood and a prior distribution in Bayesian approaches.

The gains from automatic differentiation may differ with other optimization procedures. In particular, some implementations of difference approximations start using forward differences, switching to central differences near the optimum. It is also possible that a derivative-free optimization algorithm, such as Brent's [Bren73a] modification of Powell's method, could perform well in this application.

The speed-up in the estimation of growth model parameters achieved through automatic differentiation was well worthwhile, reducing computing costs and the turn-around time for each computer run. Later, it made it feasible to perform the parameter estimation on microcomputers.

With the increased availability of inexpensive and powerful microcomputers, the preparatory work required by current automatic differentiation procedures is probably not warranted for the casual user, or for one-off or small optimization problems (see also [Soul91a]). For large problems that need to be solved repeatedly, however, automatic differentiation can be very useful.

The larger savings in computing time on VAX computers shown here agrees with our previous experience [Garc89a]. A possible reason might be differences in the degree of optimization achieved by the Fortran compilers.

In terms of storage requirements and speed, the code generated by GRAD seems fairly efficient, at least compared to JAKEF in this type of problems. Even more favorable results have been reported by Soulié in tests including also other automatic differentiation systems [Soul91a].

4 Analysis of differentiation approaches. Two techniques of automatic differentiation of algorithms are generally recognized, the *forward mode* and the *reverse mode* [Grie89a,Jued91a]. Symbolic differentiation can be used on expressions, but does not normally apply to algorithms containing loops and conditionals [Char91a]. GRAD can be considered as a hybrid approach, combining the forward mode of handling algorithmic constructs with symbolic differentiation at the statement level. Alternatively, it can be seen as extending symbolic differentiation to the manipulation of whole algorithms.

Automatic differentiation techniques are usually described and analyzed through computational trees and graphs [Rall81a,Grie89a]. A different and somewhat more general formalism is presented here, covering the use of symbolic manipulation as in GRAD.

4.1 Model of computation. Consider the execution of a subprogram or algorithm as a sequence of assignments:

$$x_{k_i} \leftarrow f_i(\mathbf{x}),\ i = 1, \ldots, m, \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ are all the variables defined in the program. In the i-th assignment the k_i-th variable is given the value of the right-hand-side of (1). The sequence of assignments may vary between runs by the action of control statements; we consider the sequence determined by a particular initial value of $\mathbf{x}$.

From a state-space point of view, the state of the process after the i-th assignment is given by the value $\mathbf{x}^i$ of the state vector $\mathbf{x}$ (the contents of the storage locations). Transitions are given by

$$\mathbf{x}^i = \mathbf{f}_i(\mathbf{x}^{i-1}) \tag{2}$$

according to the mapping

$$\begin{aligned} \mathbf{f}_i : \Re^n &\rightarrow \Re^n \\ \mathbf{x} &\mapsto (x_i, \ldots, x_{k_i-1}, f_i(\mathbf{x}), x_{k_i+1}, \ldots, x_n). \end{aligned}$$

Our subprogram must compute a function value as a function of p independent variables. Assume that the independent variables are in the first p initial values of $\mathbf{x}$:

$$(x_1^0, \ldots, x_p^0) \equiv \boldsymbol{\theta},$$

and the function value is

$$x_{k_m}^m \equiv \phi.$$

We want to compute the gradient

$$\frac{\partial \phi}{\partial \boldsymbol{\theta}} \equiv \mathbf{g}.$$

Observations:

a) All variables (storage locations) can be re-assigned.

b) At stage i, intermediate variables not initialized or yet assigned a value contain arbitrary values. In a valid program these are not used in f_i.

c) m can be much larger than the number of statements in the subprogram, for example if the subprogram contains loops.

d) We may think of (1) as assignments in a high-level language (e.g. Fortran), with arbitrary right-hand-sides. This is the view taken in GRAD. Alternatively, they may be assignments in the Assembler or machine language code after compilation, or results from elementary operations in the computational tree generated by a precompiler. This later one is the view commonly taken in the literature, with the f_i in (1) limited to primitive operations on one or two arguments.

e) I discuss only scalar assignments, as in Fortran and similar languages. The extension to languages such as APL, which allows parallel assignments to arrays or sub-arrays, is straightforward.

4.2 Forward differentiation. The forward mode is based on carrying forward the partial derivatives with respect to the independent variables:

$$\frac{\partial \mathbf{x}^i}{\partial \theta} \equiv \mathbf{Y}^i \in \Re^{n \times p}.$$

The initial value is clearly

$$\mathbf{Y}^0 = \frac{\partial \mathbf{x}^0}{\partial \theta} = [\mathbf{I}\ \mathbf{0}]^T,$$

(where $\mathbf{I}$ is an identity matrix and $\mathbf{0}$ is a matrix of zeroes) and, using the chain rule, the i-th value can be obtained as

$$\mathbf{Y}^i = \frac{\partial \mathbf{x}^i}{\partial \theta} = \frac{\partial \mathbf{x}^i}{\partial \mathbf{x}^{i-1}} \frac{\partial \mathbf{x}^{i-1}}{\partial \theta} = \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}} \mathbf{Y}^{i-1},$$

or

$$\mathbf{Y}^i = \mathbf{J}^i \mathbf{Y}^{i-1},\ i = 1, \ldots, m, \tag{3}$$

where $\mathbf{J}^i \in \Re^{n \times n}$ is the Jacobian matrix of (2). The required gradient is the k_m-th row of $\mathbf{Y}^m$.

With scalar assignments, only the k_i-th row of $\mathbf{Y}$ changes in step i. The calculations can be arranged as follows, where $\mathbf{y}_j$ denotes the j-th row of $\mathbf{Y}$:

```
$\mathbf{Y} \leftarrow [\mathbf{I}\ \mathbf{0}]^T$
for $i = 1, \ldots, m$
    $\mathbf{y}_{k_i} \leftarrow \nabla f_i \mathbf{Y}$                    (4)
    $x_{k_i} \leftarrow f_i(\mathbf{x})$
$\mathbf{g} \leftarrow \mathbf{y}_{k_m}$.
```

4.3 Reverse differentiation. The reverse mode iterates backwards carrying the partial derivatives of the final function value:

$$\frac{\partial \phi}{\partial \mathbf{x}^i} \equiv \mathbf{z}^i \in \Re^n.$$

The starting value is

$$\mathbf{z}^m = \frac{\partial \phi}{\partial \mathbf{x}^m} = \mathbf{e}_{k_m},$$

where $\mathbf{e}_j$ is the j-th Cartesian basis (row) vector. The chain rule produces the updating formula:

$$\mathbf{z}^{i-1} = \frac{\partial \phi}{\partial \mathbf{x}^{i-1}} = \frac{\partial \phi}{\partial \mathbf{x}^i}\frac{\partial \mathbf{x}^i}{\partial \mathbf{x}^{i-1}} = \mathbf{z}^i\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}}.$$

or

$$\mathbf{z}^{i-1} = \mathbf{z}^i\mathbf{J}^i,\ i = m, \ldots, 1. \tag{5}$$

The required gradient is left in the first p components of $\mathbf{z}^0$.

With scalar assignments,

$$\mathbf{J}^i = \mathbf{I} + \mathbf{e}_{k_i}^T(\nabla f_i - \mathbf{e}_{k_i}),$$

and the calculations for the "reverse sweep" are as follows:

```
$\mathbf{z} \leftarrow \mathbf{e}_{k_m}$
for $i = m, \ldots, 1$
    $\mathbf{z} \leftarrow \mathbf{z} + (\nabla f_i - \mathbf{e}_{k_i})z_{k_i}$                (6)
$\mathbf{g} \leftarrow (z_1, \ldots, z_p)$.
```

A forward sweep over the sequence (1) must be performed first to obtain the ∇f_i required in (6). There are at least two possibilities. In the immediate differentiation variant, the ∇f_i are computed and stored in the forward sweep. In the delayed differentiation variant, the $\mathbf{x}^i$ (or sufficient information to reconstruct them later) are stored in the forward sweep, and used to compute the ∇f_i as needed in the reverse sweep.

4.4 Discussion. I have ignored questions of existence and correctness for values of θ where changes in the execution sequence occur. Some careful analysis of this issue might be needed.

As described above, the computational effort in the forward and reverse strategies differs significantly only in the computation of (4) instead of (6). The expression in (4) contains np multiplications, while (6) has only n. This is the basis for the computational superiority claimed for the reverse approach [Grie89a,Hill85a].

When the sparsity of ∇f_i and $\mathbf{Y}$ is exploited, however, the situation is not that clear. It seems likely that in most instances the number of non-zeros in these arrays would grow slowly, if at all, with the number of independent variables. Properly handled, zero elements save multiplications and, in addition, some elements of the ∇f_i in (4) may not be needed at all. Although the asymptotic number of these multiplications might still be more favorable in the reverse mode, the practical significance of this is not obvious, at least with high-level assignments where the bulk of the computational effort is elsewhere.

It is clear from its description that GRAD uses a forward approach, taking advantage of sparsity to avoid unnecessary computation and storage. The multiple passes may be needed to ensure that the sparsity patterns are valid for any run-time execution sequence, and that values are correctly initialized. Unlike most (all?) other automatic differentiators, GRAD works with the high-level Fortran assignments of the input, instead of breaking them down into assignments involving just one elementary arithmetic operation or univariate function. Therefore, it may not be seen as "pure" forward mode, but rather as a hybrid , using symbolic algebraic manipulation within statements.

Symbolic differentiation has been criticized as inefficient, generating repeated common subexpressions. However, apart from the fact that the scope for generating these redundancies in typical Fortran statements is limited, it has here two important advantages. One, the resultant expressions for the derivatives are subjected to the full optimization power of the compiler. The optimization carried out by a modern Fortran compiler (including the elimination of common subexpressions) is likely to be more sophisticated than any optimization built into a differentiator. The second advantage, related to the first, is that the results from elementary operations within expressions are handled more efficiently, often in fast CPU registers, instead of having to be explicitly assigned and manipulated as program variables. In addition, the hybrid approach makes it easier to produce stand-alone code, without the overheads of a run-time support package.

JAKEF uses the (pure) reverse mode with immediate differentiation. As shown in 3.2, the storage for the ∇f_i can become prohibitive for large m. The faster run times for GRAD on the test functions may be due largely to the use of symbolic differentiation, and to the absence of support routine calls.

The storage requirements of the reverse mode could be reduced with delayed differentiation and the use of high-level statements. With a suitable data structure, it should be possible to store just the x_{k_i} and k_i. It is conceivable that reverse strategies could be advantageous for large values of p, with forward strategies preferable when $m \gg n$. The development of efficient symbolic-forward-reverse hybrids is also a possibility.

5 Conclusions.

GRAD is an effective tool for the automatic differentiation of Fortran code. It produces stand-alone code, without the overheads of run-time auxiliary packages. On the tests carried out its performance compares favorably with that of JAKEF and other available systems [Soul91a].

In its current form, GRAD might be considered only as "semi-automatic", since some manual editing of the input and/or output is still necessary. Most of this could be avoided by refinements in the implementation. It is not clear to me, however, if completely automatic, robust and "fool-proof" differentiation will ever be achievable. Experience indicates that occasionally code is generated that, although mathematically correct, is numerically unfeasible or inaccurate Some of the work in [Fisc91b] might be relevant here.

The use of automatic differentiation in parameter estimation for our growth models produced worthwhile savings in time and money. Similar results may apply to optimization and to the fitting of complex statistical models in other fields, if closely related problems need to be solved repeatedly. With the current software and cheap computing power, however, automatic differentiation may not be worth the effort for smaller or one-off problems.

The state-space view of automatic differentiation algorithms facilitates the analysis of hybrid strategies, combining the conventional forward and reverse modes with symbolic manipulation. It seems likely that hybrid procedures will often generate more efficient code than "pure" forward or reverse approaches where expressions are broken down into primitive unary and binary operations.

Appendix 1. Examples.

Two small examples of GRAD input and output are shown here. These are Problems E and F in Section 3.2.

The subroutine header in the output needs to be edited manually to change its name, and to include the derivatives in the parameters list. Note that redundant operations such as `**(2-1)` are resolved at compile time, and have no effect on runtime efficiency.

Problem E.

Input:

```
      SUBROUTINE CALFUN(X,N,FF,INF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(4)
      INF = 0
      DO 1 I = 2, 4
```

```
        IF (X(I) .GT. 0.) GOTO 1
          INF = 1
  1   CONTINUE
      IF (INF .EQ. 1) RETURN
      FF = (X(1) - 1.)**2 + (X(1) - DSQRT(X(2)))**2
      FF = FF + (X(2) - DSQRT(X(3)))**2 + (X(3) - DSQRT(X(4)))**2
      RETURN
      END
```

Output:

```
C ===  Derivating with respect to:
C     X(1) X(2) X(3) X(4)
C
      SUBROUTINE CALFUN(X,N,FF,INF)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(4)
      INF = 0
      DO 1 I = 2, 4
        IF (X(I) .GT. 0.) GOTO 1
          INF = 1
  1   CONTINUE
      IF (INF .EQ. 1) RETURN
      FF_1=2*(X(1)-1.)**(2-1)+2*(X(1)-DSQRT(X(2)))**(2-1)
      FF_2=2*(X(1)-DSQRT(X(2)))**(2-1)*(-1./2./DSQRT(X(2)))
      FF_3=0.
      FF_4=0.
      FF = (X(1) - 1.)**2 + (X(1) - DSQRT(X(2)))**2
      FF_2=FF_2+2*(X(2)-DSQRT(X(3)))**(2-1)
      FF_3=FF_3+2*(X(2)-DSQRT(X(3)))**(2-1)*(-1./2./DSQRT(X(3)))+2*(X(3)
     :-DSQRT(X(4)))**(2-1)
      FF_4=FF_4+2*(X(3)-DSQRT(X(4)))**(2-1)*(-1./2./DSQRT(X(4)))
      FF = FF + (X(2) - DSQRT(X(3)))**2 + (X(3) - DSQRT(X(4)))**2
      RETURN
      END
```

Problem F.

Input:

```
      SUBROUTINE FP (X, Y, Z, F)
C == Fletcher and Powell, 1963 ==
      IMPLICIT REAL*8 (A-H,O-Z)
      PI = 3.14159
      PI2 = 2 * PI
      R = DSQRT(X*X + Y*Y)
      IF (X .LT. 0) GO TO 10
        THTA = DATAN(Y / X) / PI2
        GO TO 20
   10 CONTINUE
        THTA = (PI + DATAN(Y / X)) / PI2
   20 CONTINUE
      F = 100. * ((Z - 10 * THTA)**2 +
     1    (R - 1)**2) + Z*Z
      RETURN
      END
```

Output:

```
C ===  Derivating with respect to:
C      X Y Z
C
      SUBROUTINE FP (X, Y, Z, F)
C == Fletcher and Powell, 1963 ==
      IMPLICIT REAL*8 (A-H,O-Z)
      PI = 3.14159
      PI2 = 2 * PI
      R_1=(X+X)/2./DSQRT(X*X+Y*Y)
      R_2=(Y+Y)/2./DSQRT(X*X+Y*Y)
      R = DSQRT(X*X + Y*Y)
      IF (X .LT. 0) GO TO 10
      THTA_1=(-Y/X)/X/(1.+(Y/X)**2)/PI2
      THTA_2=1./X/(1.+(Y/X)**2)/PI2
        THTA = DATAN(Y / X) / PI2
        GO TO 20
   10 CONTINUE
      THTA_1=(-Y/X)/X/(1.+(Y/X)**2)/PI2
      THTA_2=1./X/(1.+(Y/X)**2)/PI2
        THTA = (PI + DATAN(Y / X)) / PI2
   20 CONTINUE
      F_1=100.*(2*(Z-10*THTA)**(2-1)*(-10*THTA_1)+2*(R-1)**
          (2-1)*R_1)
      F_2=100.*(2*(Z-10*THTA)**(2-1)*(-10*THTA_2)+2*(R-1)**
          (2-1)*R_2)
      F_3=100.*2*(Z-10*THTA)**(2-1)+Z+Z
      F = 100. * ((Z - 10 * THTA)**2 +
     1    (R - 1)**2) + Z*Z
      RETURN
      END
```

Appendix 2. Application background. Foresters can influence the development of a forest stand (a homogeneous patch of forest) through a number of silvicultural treatments . Stand density (trees per hectare), can be controlled by the selection of an initial planting density, and by thinnings, which are partial cuts where usually smaller and malformed trees are removed. Density affects the total volume production. the incidence of competition-induced mortality, and the size and timber quality of individual trees. Timber quality can also be improved by pruning lower branches, possibly at some cost in reduced growth. Other management decisions may involve the application of fertilizers and pesticides. different planting or regeneration techniques. the development and use of genetically improved seed. and the timing of the final cut. Mathematical growth models capable of predicting treatment effects are essential for rational forest management. especially in intensively managed production forests.

Over the past decade. 10 regional growth models for radiata pine and one for Douglas-fir have been developed in New Zealand using a methodology based on stochastic differential equations and maximum-likelihood estimation [Garc88a]. The models can predict the behavior of stands subject to a wide range of initial densities, timing and intensity of thinnings, and in some instances pruning, fertilizing, and genetic improvement.

A brief description of these models and estimation procedures follows. More details can be found in [Garc79a,Garc84a,Garc89a], and a more general discussion of growth modeling in [Garc88b]. [Bard74a] is an excellent source for estimation theory and methods.

The state of a forest stand is assumed to be adequately described by 3 to 5 state variables: mean diameter. stand height, trees per hectare. and. in some models. measures of ground cover and/or nutrient concentrations. Treatments cause instantaneous changes in the state variables. Between treatments, the state trajectories are modeled by a system of differential equations.

The differential equations are linear on power transformations of the state variables, and can be written as

$$\frac{d\mathbf{x}^{\mathbf{C}}}{dt} = \mathbf{A}\mathbf{x}^{\mathbf{C}} + \mathbf{b},$$

defining

$$\mathbf{x}^{\mathbf{C}} = \exp[\mathbf{C} \ln \mathbf{x}].$$

$\mathbf{x}$ is the state vector, and $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{C}$ are matrices and vectors of parameters to be estimated. Some of these are actually functions, containing unknown parameters, of a site productivity index specific to each stand. Some models include additional functions of state variables multiplying the right-hand-side [Garc89a].

The data consists of a few consecutive measurements, at irregular intervals, on a large number of sample plots established in different stands. In order to devise a rational estimation procedure, the data variability is modeled as a perturbation of the differential equations by a Wiener stochastic process. The resulting stochastic differential equations can be integrated analytically to compute the likelihood function, that is, the probability of the model generating the observed data as a function of the parameters . The maximum-likelihood estimates are those values of the parameters that maximize the likelihood [Zack71a,Bard74a].

The differential equation parameters are estimated by maximum likelihood in two stages. The height growth is treated as a self-contained subsystem, since it can be assumed that the development in stand top height is approximately independent of the other state variables. Therefore, one of the equations involves only the heights, and its parameters are estimated first, together with the site index for each stand. Once these are available, the rest of the parameters are estimated using the likelihood function for the whole system. In addition to the Wiener perturbations, the height growth model also includes other random variables representing measurement errors.

The fitting of the height growth model, although simpler than the full model in being univariate, involves the optimization of functions of hundreds of variables, i. e. the different site indices and sometimes nuisance parameters (variances) for each sample plot. A full Newton algorithm, with modifications to ensure convergence, has been implemented for this purpose [Garc83a]. Although very large, the Hessian is sparse, with a special structure exploited by partitioning techniques. Analytic first and second derivatives are used, computed by a hand-coded implementation of the differentiation approach behind GRAD, described in Section 2. The procedure has proven to be very reliable and efficient.

For the rest of the parameters, the likelihood function is maximized (or rather, minus the logarithm of the likelihood is minimized) using a general-purpose variable-metric optimization routine. The smaller number of variables (9–20), and the experimental nature of the code, subject to frequent detail changes, did not justify the development of a specialized procedure as in the case of the height model.

Part IX

Automatic Differentiation Bibliography

collected by George F. Corliss

This bibliography represents the common bibliography for all of the papers in this volume. Each author prepared a bibliography for her or his own paper. The separate bibliographies were merged into a single BibTex database, and references from several other sources were added. Especially valuable contributions were made by bibliographic data bases previously compiled by Bruce Char, by David Gay and by Davis, Corliss, and Krenz [Davi88a].

This bibliography includes most of the work known to the editors in the area of automatic differentiation. Because it includes all of the works cited by any paper in this volume, it includes many citations which are not directly related to automatic differentiation. For example, it includes basic references in optimization, symbolic algebra systems, and several applications areas.

The electronic form of this bibliography contains many other works relating to automatic differentiation. The electronic version is available from netlib (`netlib@research.att.com`). Corrections and additions are welcome and should be sent to Dr. George F. Corliss, Department of Mathematics, Statistics, and Computer Science, Marquette University, Milwaukee, WI 53233 USA, georgec@boris.mscs.mu.edu.

[Abra70a] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathematical Functions*, Dover, New York, 1970.

[AdaLRM] Ada Joint Program Office, *Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A*, Washington, DC, 1983.

[Adam69a] D. S. Adamson and C. W. Winant, *A SLANG simulation of an initially strong shock wave downstream of an infinite area change*, in Proceedings of the Conference on Applications of Continuous-System Simulation Languages, 1969, pp. 231 – 240.

[AFNO83a] AFNOR, *Le langage de programmation FORTRAN*, norme ISO 1539 (norme NF Z 65-110), Association Française de Normalisation, Tour Europe, Cedex 7, F-92080 Paris La Defense Cedex, 1983.

[Ahoa86a] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1986.

References

[Bard74a] Y. BARD. *Nonlinear Parameter Estimation*. Academic Press, New York, 1974.

[Bigg71a] M. C. BIGGS, *Minimization algorithms making use of non-quadratic properties of the objective function*. Journal of the Institute of Mathematics and Its Applications, 8 (1971), pp. 315 - 327.

[Bigg73a] ——, *A note on minimization algorithms which make use of non-quadratic properties of the objective function*. Journal of the Institute of Mathematics and Its Applications, 12 (1973), pp. 337 - 338.

[Bren73a] R. P. BRENT, *Algorithms for Minimization Without Derivatives*. Prentice-Hall, Englewood Cliffs. NJ, 1973.

[Char91a] B. W. CHAR, *Computer algebra as a toolbox for program generation and manipulation*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia, PA, 1991.

[Davi81a] A. J. T. DAVIE AND R. MORRISON, *Recursive Descent Compiling*, Ellis-Horwood, Chichester, 1981.

[Fisc91b] H. FISCHER, *Special problems in automatic differentiation*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss. eds.. SIAM. Philadelphia. PA. 1991.

[Garc79a] O. GARCÍA. *Modelling stand development with stochastic differential equations*, in Mensuration for Management Planning of Exotic Forest Plantations, D. A. Elliot, ed., New Zealand Forest Service. FRI Symposium No. 20, 1979. pp. 315 - 333.

[Garc83a] ——. *A stochastic differential equation model for the height growth of forest stands*, Biometrics, 39 (1983), pp. 1059 - 1072.

[Garc84a] ——, *New class of growth models for even-aged stands: Pinus radiata in Golden Downs Forest*, New Zealand Journal of Forestry Science, 14 (1984), pp. 65 - 88.

[Garc88a] ——, *Experience with an advanced growth modelling methodology*, in Forest Growth Modelling and Prediction, A. R. Ek. S. R. Shifley, and T. E. Burk, eds., USDA Forest Service, General Technical Report NC-120, 1988, pp. 668 - 675.

[Garc88b] ——, *Growth modelling - A (re)view*, New Zealand Forestry, 33 (1988), pp. 14 - 17.

[Garc89a] ——, *Growth modelling - New developments*. in Japan and New Zealand Symposium on Forestry Management Planning, H. Nagumo and Y. Konohira, eds., Japan Association for Forestry Statistics, 1989.

[Gold91a] V. V. GOLDMAN, J. MOLENKAMP, AND J. A. VAN HULZEN, *Efficient numerical program generation and computer algebra environments*, in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM. Philadelphia. PA. 1991.

[Grie89a] A. GRIEWANK. *On automatic differentiation*, in Mathematical Programming: Recent Developments and Applications, M. Iri and K. Tanabe, eds., Kluwer Academic Publishers. 1989. pp. 83 - 108.

[Hill85a] K. E. HILLSTROM, *Users guide for JAKEF*. Technical Memorandum ANL/MCS-TM-16, Mathematics and Computer Science Division. Argonne National Laboratory, 9700 South Cass Ave., Argonne, IL 60439-4844. 1985.

[Jued91a] D. JUEDES. *A taxonomy of automatic differentiation tools*. in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss, eds., SIAM, Philadelphia. PA. 1991.

[Lill70a] S. A. LILL. *Algorithm 46: A modified Davidon method for finding the minimum of a function using difference approximation for derivatives*, The Computer Journal, 13, 14 (1970). pp. 111 - 113, 106.

[NOC76a] N.O.C.. *OPTIMA - Routines for optimisation problems*. technical report. The Numerical Optimisation Center. Hatfield Polytechnic. Hatfield. UK, 1976.

[ONei71a] R. O'NEILL. *Algorithm AS47 - Function minimization using a simplex procedure*. Applied Statistics. 20. 23. 25 (1971). pp. 338 – 346. 250 – 252. 97.

[Rall81a] L. B. RALL. *Automatic Differentiation: Techniques and Applications*. vol. 120 of Lecture Notes in Computer Science, Springer-Verlag. Berlin. 1981.

[Soul91a] E. SOULIÉ. *User's experience with FORTRAN precompilers for least squares optimization problems*. in Automatic Differentiation of Algorithms: Theory, Implementation, and Application, A. Griewank and G. F. Corliss. eds., SIAM, Philadelphia, PA. 1991.

[STSC89a] STSC. *APL*PLUS System for the PC - User's Manual*, STSC, Inc.. Rockville. Maryland. 1988.

[Turn85a] J. R. TURNER. *Pocket APL - Reference Guide*. STSC, Inc., Rockville. Maryland. 1985.

[Weng64a] R. E. WENGERT, *A simple automatic derivative evaluation program*. Comm. ACM. 7 (1964). pp. 463 – 464.

[Zack71a] S. ZACKS. *The Theory of Statistical Inference*. John Wiley and Sons. New York. 1971.