A TUTORIAL ON SIMPLICITY AND COMPUTATIONAL DIFFERENTIATION FOR STATISTICIANS

Stephen P. Smith
Visiting Scientist
UC Davis Physics Department
Davis, CA

Abstract. Automatic differentiation is a powerful collection of software tools that are invaluable in many areas including statistical computing. It is well known that automatic differentiation techniques can be applied directly by a programmer in a process called hand coding. However, the advantages of hand coding with certain applications are less appreciated, but these advantages are of paramount importance to statistics in particular. Based on the present literature, the variance component problem using restricted maximum likelihood is an example where hand coding derivatives was very useful relative to automatic or algebraic approaches. Some guidelines for hand coding backward derivatives are also provided, and emphasis is given to techniques for reducing space complexity and computing second derivatives.

Key words. Automatic differentiation, backward differentiation, Cholesky decomposition, restricted maximum likelihood.

1. Introduction

A clear understanding of numerical algorithms is sometimes very important to statisticians for both theoretical and applied reasons. The sweep operator (Goodnight 1979), and the eigenvector and eigenvalue computation (Dempster, Selwyn, Patel and Roth 1984) are examples of numerical operations that have been explored by statisticians. In the same tradition, this paper adds automatic differentiation (AD) to the list of numerical tools that statisticians should understand, and it is done in context of derivative computation for purpose of maximum likelihood estimation (as it relates to linear models and variance component estimation).

More generally, AD represent a collection of software tools that are used to calculate derivatives. It is finding application in many areas, not just statistics, but also including optimization, dynamical systems, weather modeling, engineering and control theory. The typical situation involves calculation of derivatives of a function, or a collection of functions, that are represented by lines of computer code. The automatic differentiation of the computer code avoids the tedium of direct programming of the derivatives. The benefits of AD are most apparent when the source code is very large and when it is too costly to redesign the software from scratch.

This paper is very much about simplicity, and human insight and intervention, and how these relate to AD. The following section provides an introduction to AD as it is commonly known, and simplicity is treated in subsequent sections.

2. Automatic Differentiation

Automatic differentiation involves the evaluation of a derivative, or set of derivatives, by applying the chain rule of calculous directly to the steps of an algorithm. These steps make up a recursion list, and Griewank (1992) described differentiation as a series of matrix multiplications, $\mathbf{A}_1 \times \mathbf{A}_2 \times \cdots \times \mathbf{A}_k$, where a multiplication involving $\mathbf{A}_i$ applies the associated partial derivatives of the i-th recursion as permitted by the chain rule. There are two main types of AD depending on how the chain rule is applied, as indicated by the order of these matrix multiplications. If the chain rule is applied in a forward mode, from input parameters to the final output of a function, the process is called forward differentiation. The matrices are multiplied in the order they are generated. If the chain rule is applied by reversing the order matrices are multiplied, the process is called backward differentiation. In backward differentiation, the derivatives of the last function (with respect to the intermediate calculations) are accumulated until they relate to the input parameters. This permits the calculation of the entire gradient vector with one reverse application of the chain rule, but this remarkable calculation is done with possible increased storage requirements (or added complexity to treat the storage needs) because the program needs to retrieve the intermediate calculations going backward.

Automatic differentiation depends on predefined parameters and the resulting intermediate calculations, and therefore, the result of AD are real numbers or an algorithm to compute them. While AD can be represented symbolically, it is important to understand that AD is not symbolic differentiation as found in early versions of some popular software packages like S, Mathematica or Maple. The main difference is that symbolic differentiation is applied to an expression and it results in a new expression, whereas AD takes an algorithm and turns it into a new algorithm. As a result, symbolic differentiation of some expressions is difficult because of combinatorial explosion of symbolism, but this problem is not a quality of AD. Moreover, AD is not a numerical method to approximate derivatives using finite differences. Aside from rounding errors, AD gives exact derivatives. Forward differentiation requires the same work as numerical derivatives derived from finite differences, whereas backward differentiation can provide a significantly faster calculation of a gradient vector than what is achievable from either numerical differentiation or forward differentiation.

The literature on AD is vast, but it is disconnected due to the variety of areas where it has been used. Two conference proceedings (Griewank and Corliss 1991; Berz, Bischof, Corliss and Griewank 1996), provide good surveys of the existing methods and software tools. The papers by Baur and Strassen (1983) and Griewank (1989) are most insightful. While it is surprising how little of AD has been disseminated to applied scientists, the popularity of the tools has been expanding steadily since the first conference in 1991. The most recent advances in AD are found in a text book by Griewank (2000).

One possible explanation of the slow introduction of AD is that it is sometimes treated as a "black box". That is, source code is presented to the black box and by some miraculous magic the derivative code emerges. In this way the end users do not toil with the details hidden in the box. While this is more positive than negative, a regrettable side effect is that users may not develop a deep appreciation of automatic differentiation. The black box is not promoted because it is not understood.

In this paper, we describe AD as a collection of software tools that can be used directly by programers, and we also present the opposing view and point to cases where the automatic treatment is more appropriate. While this paper is not intended to slight AD, we would like to suggest that human insight deserves promotion. Moreover, the popularity of AD should benefit when the techniques are understood more clearly by applied scientists that enjoy programing

3. Guidelines for Human Programmers

The approach to treat AD as a set of ideas that can be used while developing derivative code directly is called hand coding, and this requires a thorough understanding of both forward and backward differentiation. Even if a software tool is used, some hand coding may be required for treating exceptions or to improve efficiency (Hovland, Bischof, Spiegelman and Casella 1997). While this use of computational differentiation promotes understanding, large-scale hand coding is not acceptable for the differentiation of a large computer program. Hand coding is useful for small or simple computer programs or subroutines, and while hand coding has enjoyed an obvious presence in application of AD (Christian, Davies, Dixon, Roy, and Van Der Zee 1997), hand coding has not been actively promoted. This is abundantly clear from the unfortunate use of the word "automatic", and a resistence by educators to provide details like those presented below. The prejudice may be due to the false beliefs that small or simple programs can be treated adequately by analytic or symbolic methods, and that there is no big advantage from hand coding of forward or backward derivatives in these cases. It sometimes happens that the hand coding approach may produce useful results when available AD tools can't be used efficiently.

3.1 Representing the Algorithm as a Recursion List

A useful notation for the algorithm is the recursion list given below.

$h_1 \leftarrow f_1(\Omega_1)$
$h_2 \leftarrow f_2(\Omega_2)$
.
.

$$h_k \leftarrow f_k(\Omega_k)$$

where Ω_r is a subset of all the intermediate variables prior to step r in the algorithm, and f_r is the transforming recursion. Mathematically, we write $\Omega_r \subseteq \mathbf{x} \cup \{h_i\colon i<r\}$ and $\mathbf{x}$ is a parameter set of dimension n. But in general, $h_1=x_1$, $h_2=x_2$, ... $h_n=x_n$, so it may be assumed that $\Omega_r \subseteq \{h_i\colon i<r\}$ for $r>n$. The objective of differentiation is to write an algorithm for computing $\partial h_k/\partial \mathbf{x}$. The intermediates, h_i, i=1, 2, ... k, are assumed to be scalar values, which is the optimum situation for backward differentiation. While vector-valued intermediates can be permitted with a more complex notation, the above representation can be forced. Forcing this representation has no effect on the formulae of Sections 3.2 and 3.4 when human insight is used to define partial derivatives of f_i, i=1,2, ... k. The intermediates can still be computed by the most frugal algorithm when the above representation is less than efficient.

The recursion list need not represent elementary operations. Each of the operations, f_i, i=1,2, ... k, may be relatively complex, to reflect a high level of granularity (Bischof and Haghighat 1996). In the case of backward differentiation, the optimum level of granularity is a balance between how easy it is for the programmer to differentiate the f_i versus how much storage the programmer wishes to use for the intermediates.

The focus of the remainder of this paper will be on backward differentiation. While forward differentiation will not be described (is less problematic), it is important to appreciate that it can be very useful for parts of the recursion list, particularly when some intermediates are vector-valued (Bischof and Haghighat 1996; Hovland, et. al. 1997).

3.2 Backward Differentiation

As indicated, backward differentiation starts with the last recursion, i.e., $h_k=f_k(\Omega_k)$, and proceeds backward through the recursion list. The chain rule is applied at each step, while accumulating the derivatives, $\partial h_k/\partial h_i$ $(i<k)$ in the vector **g**. Eventually the derivatives are related back to the original parameters, **x**. This provides the calculation of the entire gradient vector with a single reverse pass through the recursions:

1. Set $g_i=0$ for $i<k$, and $g_k=1$.
2. $g_i \leftarrow g_i + g_r\, \partial f_r/\partial h_i$ for all i where $h_i \in \Omega_r$, $r=k, k-1, ..., n+1$.
3. Then $\partial h_k/\partial x_i = g_i$, $i=1, 2, ... n$, and, in general, $\partial h_k/\partial h_i = g_i$, $i=1, 2, ... k$.

Vendors of AD may call the human implementation of these instructions tedious and error prone, but these qualities are only applicable to specific cases. The fact that backward differentiation can be casts as an ordered series of matrix multiplications may conceivably confuse programmers. But the above description has a deliberate structure that is hard to get wrong. Note in particular, the correspondence in indexes used for h_i and g_i. This does not necessarily mean that g_i is physically saved at the i-th position in **g**. But it is a useful restriction of sorts, because it permits a programmer to set up a vector **h** containing h_i at location a(i). The programmer is then permitted to store g_i at position a(i) in **g**, or use some other handy addressing function of i. Much effort can be invested in the function a(i) to make its use very efficient, as in the case with sparse-matrix operations. And hand coding permits the deliberate use of these same highly optimized tools with backward differentiation without being at the mercy of software. Furthermore, the correspondence between h_i and g_i plays a key role in overwriting, as emphasized in the following section.

3.3 Reducing Space Complexity

The major storage requirement for backward differentiation is for the vectors **h** and **g**. These vectors are proportional to the number of intermediates, and with low granularity this implies that the storage requirement will be proportional to the run-time. Hence, devices are needed to reduce the storage needs. Approaches for reducing space complexity includes check-pointing and even attempts to reverse the calculations (Griewank 1992). But what works best for hand coding is increasing granularity, and/or utilizing overwriting. The effect of granularity is already apparent, so in this section only overwriting is described.

To lend itself to overwriting, the algorithm must have two qualities: (1) the recursion list for h_i , i=1,2, ... k, must show overwriting, with **h** having fewer than k elements; and (2) it must be possible to regenerate the partial derivatives for all the operations f_i, i=1, 2, ... k, from the vector **h** after the forward sweep. The first requirement implies that some h_j is lost going forward at some step m ($j<m$), and this implies that h_j is never needed in the algorithm for all steps $>m$. What this means going backward is that g_j is not needed until steps $\leq$ m. Since, going backward, g_m is needed only for steps $\geq$m, g_j can be initialized and used in place of g_m in **g** just after step m.

But to get the procedure in Section 3.2 to work, the second requirement must also hold.

The common recursions that lead to overwriting are operations like $h_m=h_j+f(\{\Omega_m - h_j\})$ where a(m)=a(j), i.e., intermediates that are lost only affect the recursions in an additive way where all the respective partials equal 1. During the reverse sweep, at step m, we have the trivial assignment $g_j \leftarrow g_m$ which can be ignored in implementation because a(m)=a(j).

A good illustration of the overwriting feature is presented in Section 4.2, where Cholesky algorithm is differentiated while using the lowest possible granularity. It is also possible to rearrange the Cholesky recursions there by permitting an increase in granularity, and accomplish an almost identical feat without using any overwriting.

3.4 Second Derivatives

Second derivatives are calculated by applying automatic differentiation twice. This can consist of two rounds of forward differentiation, or forward differentiation applied to the backward equations, or backward differentiation applied to the forward equations. But the most tedious and error prone combination, a double application of backward differentiation, will be demonstrated in this section.

To define the recursion list, the recursions of Sections 3.1 and 3.2 must be concatenated to form a larger algorithm. This algorithm is represented by the intermediates contained in the vectors **h** and **g**, and to these vectors assign the vectors **s** and **q** for accumulating derivatives. This assignment is given symbolically by

$$\begin{bmatrix} \boldsymbol{h} \\ \boldsymbol{g} \end{bmatrix} \leftrightarrow \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{q} \end{bmatrix}$$

Partial derivatives are accumulated in the vectors **q** and **s** by progressing through the concatenated algorithm in reverse order, just as before. But note that the overwriting principle in Section 3.3 applies to the recursions representing **g**; see Step 2 in Section 3.2. In fact, **g** and **q** are the same length as **h** and **s**, and this dimension is less than the number of intermediates involved in the backward sweep of Section 3.2. These simplifications produce the following algorithm for second derivatives:

```
1. Set $q_v=1$ for some $v\in\{1,2, ... n\}$, and $q_i=0$ for $i\neq v$. Set $s_j=0$ for all j.
2.      $q_r \leftarrow q_r+q_i\partial f_r/\partial h_i$, for all i where $h_i\in\Omega_r$;
        $s_j \leftarrow s_j+q_i\, g_r\partial^2 f_r/\partial h_i\partial h_j$, for all i and j where $h_i,h_j\in\Omega_r$;
        $r=n+1, n+2, ..., k$.
3. $s_i \leftarrow s_i + s_r\partial f_r/\partial h_i$ for all i where $h_i\in\Omega_r$, $r=k, k-1, ..., n+1$.
4. Then $\partial^2 h_k/\partial x_v\partial x_i = s_i$, $i=1, 2, ..., n$.
```

A few features are noteworthy. Firstly, Step 3 is identical to Step 2 in Section 3.2. Secondly, the calculation of **q** is identical to forward differentiation of the recursion list with respect to x_v. It is easy to adapt the calculations for directional derivatives by initializing **q** appropriately in Step 1. Hence, the scheme is suitable for optimization techniques that use a small number of second directional derivatives.

4. Some Experiences with Variance Component Estimation

4.1 Historical Perspective

Examples are described in this section where hand coding was found very useful, even in the presence of analytic derivatives and AD software packages. These are among the variance estimation problems that were thought to be intractable prior to 1993, partly because few knew of backward differentiation. Variance component estimation has been a very active area of investigation, and remarkably, there were many missed opportunities to re-invent backward differentiation. Our failure to appreciate backward differentiation in treating these popular problems is additional testimony of the slow dissemination of AD.

The variance component problems are related to the mixed linear model given by

$\mathbf{y}=\mathbf{X}\beta+\mathbf{Z}\mathbf{u}+\mathbf{e}$

where $\mathbf{y}$ is a vector of observations, β a vector of unknown fixed effects, $\mathbf{u}$ is a vector of random effects, and $\mathbf{e}$ is a vector of random residuals. The $\mathbf{X}$ and $\mathbf{Z}$ are known incident matrices that assign the various effects to observations. The mean of the random effects is null, and the variance-covariance structure is defined by var$\{\mathbf{u}\}=\mathbf{G}$ and var$\{\mathbf{e}\}=\mathbf{R}$. Therefore, the mean of $\mathbf{y}$ is $\mathbf{X}\beta$ and its variance matrix is $\mathbf{V}=\mathbf{Z}\mathbf{G}\mathbf{Z}'+\mathbf{R}$. The best linear unbiased estimate of β and the best linear unbiased prediction of $\mathbf{u}$ are obtained by solving the mixed model equations (Henderson, Kempthorne, Searle and Von Krosigk 1959):

$$\begin{bmatrix} X'R^{-1}X & X'R^{-1}Z \\ Z'R^{-1}X & Z'R^{-1}Z+G^{-1} \end{bmatrix} \begin{bmatrix} \hat{\beta} \\ \hat{u} \end{bmatrix} = \begin{bmatrix} X'R^{-1}y \\ Z'R^{-1}y \end{bmatrix} \qquad (4.1)$$

The mixed model equations depend on **G** and **R**, and estimates of these matrices are required if a solution to (4.1) is sought. More precisely, **G** and **R** are typically functions of unknown variances and covariances, and the estimation of these unknown parameters is called the variance component problem.

One approach used to estimate the variance components is maximum likelihood. In the case of multivariate normality, the log-likelihood is

$$\text{const.} - \tfrac{1}{2}\log|\mathbf{V}| - \tfrac{1}{2}(\mathbf{y}-\mathbf{X}\beta)'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\beta) \qquad (4.2)$$

The estimates of β and the variance components (**R** and **G**) that maximize (4.2) are the maximum likelihood estimates. But these estimates of the variance components can be seriously biased by small-sample errors related to the estimation of β. To treat this situation, Patterson and Thompson (1971) proposed restricted maximum likelihood (REML), where variance components are found by maximizing

$$\text{const.} - \tfrac{1}{2}\log|\mathbf{V}| - \tfrac{1}{2}\log|\mathbf{X}'\mathbf{V}\mathbf{X}| - \tfrac{1}{2}(\mathbf{y}-\mathbf{X}\hat{\beta})'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{X}\hat{\beta}) \qquad (4.3)$$

where $\hat{\beta}$ is given from Equation (4.1). Harville (1977) described (4.3) as a likelihood function for error contrasts, i.e., all linear combinations of **y** that are invariant to β. An equivalent but different form of the likelihood is usually more suitable for numerical computation. To construct this likelihood, follow Laird's (1982) argument and treat β as random with mean vector μ_β and variance matrix $\alpha\mathbf{I}$. This prior distribution becomes noninformative as $\alpha\to\infty$, and this is required to treat β as fixed in the Bayesian context. Therefore, the mean and variance of **y** are:

$E\{\mathbf{y}\}=\mathbf{X}\mu_\beta$

$\mathrm{var}\{\mathbf{y}\}=\mathbf{V}_\alpha= \mathbf{W}\mathbf{Q}_\alpha\mathbf{W}'+\mathbf{R}$

where

$$\boldsymbol{W}=\begin{bmatrix}\boldsymbol{X} & \boldsymbol{Z}\end{bmatrix}\ ,\quad \boldsymbol{Q}_{\boldsymbol{\alpha}}=\begin{bmatrix}\alpha\boldsymbol{I} & \mathbf{0}\\ \mathbf{0} & \boldsymbol{G}\end{bmatrix}$$

These can be substituted into the multivariate normal likelihood, while using expressions for the determinant and inverse of a matrix sum (Henderson and Searle 1981),

$$|\mathbf{V}_\alpha|=|\alpha\mathbf{I}|\cdot|\mathbf{G}|\cdot|\mathbf{R}|\cdot|\mathbf{W}'\mathbf{R}^{-1}\mathbf{W}+\mathbf{Q}_\alpha^{-1}|$$

$$\mathbf{V}_\alpha^{-1}= \mathbf{R}^{-1}-\mathbf{R}^{-1}\mathbf{W}(\mathbf{W}'\mathbf{R}^{-1}\mathbf{W}+\mathbf{Q}_\alpha^{-1})^{-1}\mathbf{W}'\mathbf{R}^{-1}$$

By letting $\alpha\to\infty$, the relevant part of the log-likelihood becomes

$$\text{const.} -\tfrac{1}{2}\log|\mathbf{R}| -\tfrac{1}{2}\log|\mathbf{G}| -\tfrac{1}{2}\log|\mathbf{C}| -\tfrac{1}{2}\mathbf{y}'\mathbf{P}\mathbf{y} \qquad (4.4)$$

where $\mathbf{C}=\mathbf{W}'\mathbf{R}^{-1}\mathbf{W}+\mathbf{Q}_\infty^{-1}$ is the coefficient matrix of the mixed model equation (4.1), and $\mathbf{P}=\mathbf{V}_\infty^{-1}$. Note that (4.4) is independent of both β and μ_β.

Aside from the simple cases involving balanced designs, the major difficulty in maximizing

(4.2), (4.3) and (4.4) had been due to the need to calculate first and second derivatives. Harville (1977) described this difficulty, and related it to the work required by several algorithms that use various amounts of information from derivatives.

Approaches to simplify derivative calculation have been actively pursued by researchers, even before AD was understood by many. The W transformation was one of the most important discoveries (Goodnight and Hemmerle 1979; Hemmerle and Hartley 1973), and more recently it has been imitated by a factorization technique for symmetric and indefinite matrices (Fraley and Burns 1995). Originally the W transformation was a device to reduce matrix computation related to derivatives of (4.2) or (4.3). But for REML, the benefits of this tool is already realized in the form of log-likelihood (4.4). However, even with (4.4) the task was thought to be impractical at one time, and required the inverse of large matrices as can be seen by symbolic differentiation of two of the most computationally expensive terms in (4.4), i.e., log|**C**| and **y'Py**:

$$\partial \log|\mathbf{C}|/\partial w = \mathrm{tr}[\mathbf{C}^{-1}\cdot\partial\mathbf{C}/\partial w] \tag{4.5}$$

$$\partial^2 \log|\mathbf{C}|/\partial w\partial v = \mathrm{tr}[\mathbf{C}^{-1}\cdot\partial^2\mathbf{C}/\partial w\partial v] - \mathrm{tr}[\mathbf{C}^{-1}\cdot\partial\mathbf{C}/\partial v\cdot\ \mathbf{C}^{-1}\cdot\partial\mathbf{C}/\partial w] \tag{4.6}$$

$$\partial\mathbf{y'Py}/\partial w = -\mathbf{y'P}\cdot\partial\mathbf{V}/\partial w\cdot\mathbf{Py} \tag{4.7}$$

$$\partial^2\mathbf{y'Py}/\partial w\partial v = 2\mathbf{y'P}\cdot\partial\mathbf{V}/\partial v\cdot\mathbf{P}\cdot\partial\mathbf{V}/\partial w\cdot\mathbf{Py} - \mathbf{y'P}\cdot\partial^2\mathbf{V}/\partial w\partial v\cdot\mathbf{Py} \tag{4.8}$$

Because **Py** equates to predictions of the residuals (multiplied by $\mathbf{R}^{-1}$) in the mixed linear model, derivatives of **y'Py** are easy to derive and compute. While derivatives of log|**C**| have been harder to compute than those for **y'Py**, simplifications were also found for log|**C**| when **C** is large and sparse (Feller 1987; Tier and Smith 1989) or for the situation of repeated measures and nested models (Giesbrecht 1978; Jennrich and Schluchter 1986; Laird, Lange and Stram 1987; Lindstorm and Bates 1988). Moreover, special structures of **C** can sometimes be utilized to ease the burden of computation (Dempster, et. al. 1984; Harville and Callanan 1990; Jensen and Mao 1988; Meyer 1985; Smith and Graser 1986). Unfortunately, analytic derivatives were still too hard to compute for the general case, and Graser, Smith and Tier (1987) proposed an algorithm that did not use derivatives and was based on the mixed model matrix:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{X'R^{-1}X} & \boldsymbol{X'R^{-1}Z} & \boldsymbol{X'R^{-1}y} \\ \boldsymbol{Z'R^{-1}X} & \boldsymbol{Z'R^{-1}Z+G^{-1}} & \boldsymbol{Z'R^{-1}y} \\ \boldsymbol{y'R^{-1}X} & \boldsymbol{y'R^{-1}Z} & \boldsymbol{y'R^{-1}y} \end{bmatrix}$$

While **M** is usually very large, it is also sparse and its Cholesky decomposition ($\mathbf{LL'}=\mathbf{M}$, $\mathbf{L}_{N\times N}$ lower triangular) is typically easy to compute using sparse-matrix methods. Components of (4.4) can be computed by noting that $f(\mathbf{L}) = -½\log|\mathbf{C}| - ½\mathbf{y'Py} = -½L_{NN}^2 - \Sigma_{i<N}\log(L_{ii})$ where L_{ij} is the ij*th* element of **L**, provided only that the first N−1 rows and columns of **M** are rearranged to make **L** more sparse. This revelation paved the way for derivative-free algorithms (Boldman and Van Vleck 1991; Meyer 1989; Meyer 1991), but surprisingly forward and backward differentiation of f(**L**) was not developed until later. Meanwhile, methods that require derivative computation were made feasible by advances in high-speed computing where AD was not used (Misztal 1990; Searle 1989; Wolfinger, Tobias and Sall 1994).

Smith (1995) showed how to compute first and second derivatives of f(**L**) by backward differentiation. This was done in real space as an extension of the sparse Cholesky factorization, and it was implemented with near optimal run-time performance. The algorithmic approach avoided all the heavy algebra of the past. This is not to say that algebraic manipulation is of no value. Simplification of (4.8) can yield an algorithm that is as good as or better than backward differentiation, and it can be used to approximate second derivatives of (4.4) (Johnson and Thompson 1995; Meyer 1997). But it is important to note that equations (4.5) to (4.8) are not algorithms, and attempts to manipulate these equations analytically did not regenerate forward and backward derivatives except in the accidental case involving the sparse inverse described in Section 4.2. History shows that direct manipulation of (4.5) and (4.6) frequently produced complex algorithms that were difficult to use.

4.2 Differentiation of Cholesky's Algorithm and the Sparse Inverse

In this section we will illustrate the use of backward differentiation to calculate (4.5), and we start with an algorithm for computing log$|\mathbf{C}_{N\times N}|$. Let **L** be the Cholesky decomposition of **C**, i.e., $\mathbf{LL}'=\mathbf{C}$. To evaluate log$|\mathbf{C}|$, therefore, first set $L_{ij}=C_{ij}$ $(i\geq j)$, and apply the recursions:

```
k=1 to N
        $L_{kk} \leftarrow L_{kk}^{1/2}$
        $L_{jk} \leftarrow L_{jk}/L_{kk}$, j=k+1, ..., N.
        $L_{ij} \leftarrow L_{ij} - L_{ik}L_{jk}$, j=k+1, ...N, i=j, ...N.
end k
```

Then $\log|\mathbf{C}|=2\sum_i \log(L_{ii})$.

These recursions can be backward differentiated by the rules of Section 3.2. However, one must be careful to note that different intermediate calculations are represented by common symbolism. At the same time one appeals to the overwriting principals of Section 3.3, and with minor simplification we derive the following algorithm that involves a half-stored matrix **G**.

Set $\mathbf{G}=\text{diag}\{2/L_{11}, 2/L_{22}, ..., 2/L_{NN}\}$ and apply the following recursions for k decreasing from N to 1.

```
    j=k+1 to N, i=j to N
        $G_{ik} \leftarrow G_{ik} - G_{ij} \times L_{jk}$
        $G_{jk} \leftarrow G_{jk} - G_{ij} \times L_{ik}$
    end i; end j

    j=k+1 to N
        $G_{jk} \leftarrow G_{jk}/L_{kk}$
        $G_{kk} \leftarrow G_{kk} - G_{jk} \times L_{jk}$
    end j

    $G_{kk} \leftarrow \frac{1}{2} G_{kk}/L_{kk}$
```

Only at the end must the parameter w be specified to compute (4.5) as

$$\partial \log|\mathbf{C}|/\partial w = \sum_{i \geq j} G_{ij} \cdot \partial C_{ij}/\partial w,$$

and hence the entire gradient vector is provided cheaply at this point. Alternatively, Misztal and Perez-Encisco (1993) showed how to evaluate (4.5) using the sparse-matrix inverse of Takahashi, Fagan and Chen (1973). Not only was this approach surprisingly economical, it represents a rare example where symbolic or algebraic analysis works well and has so been adopted in specialized software (Neumaier and Groeneveld 1998).

We now show that the sparse-inverse approach to evaluate (4.5) can be retrieved from our above algorithm with minor substitution (showing their algorithmic equivalence), and this is despite the fact that Takahashi's derivation was unrelated to differentiation. Arranging the recursions of the Cholesky decomposition as done above creates the side effect that $\{G_{kk}, G_{k+1,k}, ..., G_{Nk}\}$ are computed in isolation to other elements of **G**, at step k. Therefore, the step k calculations can be expressed with simple summations, and furthermore, the divisions involving L_{kk} can be pulled inside the respective summations to get the following. For k decreasing from N to 1:

$$G_{jk} = -G_{jj}\frac{L_{jk}}{L_{kk}} - \sum_{i=k+1}^{N} G_{ji}\frac{L_{ik}}{L_{kk}} \text{ , } j=k+1,...,N$$
$$G_{kk} = \frac{1}{L_{kk}^{2}} - ½ \sum_{j=k+1}^{N} G_{jk}\frac{L_{jk}}{L_{kk}}$$

where $G_{ji}=G_{ij}$ for the cases where j<i. Note that $G_{jj}L_{jk}/L_{kk}$ occurs at two places in the top equation.

For purposes of substitution, let **U** be unit lower triangular and **D** diagonal such that **UDU**'=**C**. Therefore, **L**=**UD**^{½}. The equations are expressed below in a more economical form. For k decreasing from N to 1:

$$G_{jk} = -G_{jj}U_{jk} - \sum_{i=k+1}^{N} G_{ji}U_{ik} \text{ , } j=k+1,...,N$$
$$G_{kk} = \frac{1}{D_{kk}} - ½ \sum_{j=k+1}^{N} G_{jk}U_{jk}$$

This still does not look quite like the sparse inverse. However, the sparse inverse represent a symmetric matrix, whereas, the above represents derivatives involving intermediates from a half-stored matrix. To connect the two, make one more substitution:

$$Z_{ij} = \begin{cases} G_{ii} \text{ if } i=j \\ \\ ½G_{ij} \text{ if } i \neq j \end{cases}$$

The following algorithm is retrieved, and it is identical to the calculation of Takahashi's sparse inverse given as **Z**. For k decreasing form N to 1:

$$Z_{jk} = -\sum_{i=k+1}^{N} Z_{ji} U_{ik} \text{ , } j=k+1,\ldots,N$$
$$Z_{kk} = \frac{1}{D_{kk}} - \sum_{j=k+1}^{N} Z_{jk} U_{jk}$$

It is easy to verify that the above algorithmic equivalence between backward differentiation and Takahashi's sparse inverse is maintained when sparse structures are encountered. In which case either methods can be used to evaluate (4.5), and avoidance can be made of the cubic work and quadratic storage that would characterize full matrix inversion.

5. The Case for Automatic Tools

The complexity of REML is reduced because it relates to a simple algorithm involving the Cholesky decomposition. The Cholesky decomposition only has three lines of computer code that involve floating-point arithmetic. This makes forward and backward differentiation very easy to develop and optimize by hand. But if consideration is to be given to AD, an appreciation of purpose is required. While the Cholesky decomposition is a simple process, the REML applications are, nevertheless, computer intensive. Additionally, REML applications are repetitive enough to justify the human effort involved in writing efficient programs. Therefore, efficiency is more important than ease of use, and this minimizes the main advantage that AD tools have over hand coding. Nevertheless, ADIFOR is a very useful tool for evaluating forward derivatives automatically (Bischof and Carle 1996), including forward derivatives of (4.4). But backward derivatives are more useful than forward derivatives for typical REML applications (Meyer and Smith 1996), and to compute backward derivatives automatically a tool like Odyssée is required (Mohammadi, Male and Rostaing-Schmidt 1996). Attempts to compute backward derivatives by applying an AD tool to REML code might add considerable costs, because of possible check-pointing (Griewank 1992).

Because AD is most useful with complex problems, AD and hand coding complement each other. The search for good illustrations of AD necessarily leads to complexity, just the opposite of hand coding. Variance component estimation is almost too simple to provide a favorable illustration of AD. But AD works best as a tool to calculate derivatives for the most troublesome likelihood functions, like the likelihood for the rank-regression model given by Smith and Hammond (1988). Independent of the issue of complexity, one does find application for automatic calculation with nonlinear models (Bates and Chambers, 1992).

REFERENCES

Baur, W., and Strassen, V. (1983), The complexity of partial derivatives, Theoretical Computer Science, 22, 317-330.

Bates, D.M., and Chambers, J.M. (1992), Nonlinear models, in Statistical Models in S, eds. J. M.

Chambers, and T.J. Hastie, Pacific Grove, CA: Wadsworth & Brooks/Cole, Ch. 10.

Berz, M., Bischof, C., Corliss, G., and Griewank, A., eds. (1996), Computational differentiation: Techniques, applications, and tools, Philadelphia: SIAM.

Bischof, C.H., and Carle, A. (1996) User's experience with ADIFOR 2.0 , in Computational differentiation: Techniques, applications, and tools, eds. M. Berz, C. Bischof, G. Corliss, and A. Griewank, Philadelphia: SIAM, pp. 385-392.

Bischof, C.H., and Haghighat, M.R. (1996), Hierarchical approaches to automatic differentiation, in Computational differentiation: Techniques, applications, and tools, eds. M. Berz, C. Bischof, G. Corliss, and A. Griewank, Philadelphia: SIAM, pp. 83-94.

Boldman, K.G., and Van Vleck, L.D. (1991), Derivative-free restricted maximum likelihood estimation in animal models with a sparse matrix solver, Journal of Dairy Science, 74, 4337-4343.

Christianson, D.B.,Davies, A.J., Dixon, L.C.W., Roy, R., and Van Der Zee, P., (1997), Giving reverse differentiation a helping hand, Optimization Methods and Software, 8, 53-67.

Dempster, A.P., Selwyn, M.R., Patel, C.M., and Roth, A.J. (1984), Statistical and computational Aspects of mixed model analysis, Applied Statistics, 33, 203-214.

Fellner, W.H. (1987), Sparse matrices and the analysis of variance components by likelihood methods, Communication in Statistics and Simulation, 16, 439-463.

Fraley, C., and Burns, P.J. (1995), Large-scale estimation of variance and covariance components, SIAM Journal on Scientific Computing, 16, 192-209.

Giesbrecht, F.G. (1978), Estimating variance components in hierarchical structures using MINQUE and restricted maximum likelihood, Communications in Statistics: A - Theory and Methods, 7, 891-904.

Goodnight, J.H. (1979), A tutorial on the sweep operator, The American Statistician, 33, 149-158.

Goodnight, J.H., and Hemmerle, W.J. (1979), A simplified algorithm for the W-transformation in variance component estimation, Technometrics, 21, 265-268.

Graser, H.-U., Smith, S.P., and Tier, B. (1987), A derivative free approach for estimating variance components in animal models by REML, Journal of Animal Science, 64, 1362-1370.

Griewank, A. (1989), On automatic differentiation, in Mathematical Programming: Recent Developments and Applications, eds. M. Iri and K. Tanabe, Kluwer Academic Publishers, Dordrecht, pp. 83-108.

Griewank, A., and Corliss, G.F., eds. (1991), Automatic differentiation of algorithms: theory, implementation, and application, Philadelphia: SIAM.

Griewank, A., (1992), Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, Optimization Methods and Software, 1, 35-54.

Griewank, A., (2000), Evaluating derivatives: principles and techniques of algorithmic differentiation, Philadelphia, SIAM.

Harville, D.A. (1977), Maximum likelihood approaches to variance component estimation and to related problems, Journal of the American Statistical Association, 72, 320-340.

Harville, D.A., and Callanan, T.P. (1990), Computational aspects of likelihood-based inference for variance components, in Statistical Methods for Genetic Improvement of Livestock, eds. D. Gianola and K. Hammond, New York: Springer-Verlag, pp. 136-176.

Hemmerle, W.J., and Hartley, H.O. (1973), Computing maximum likelihood estiamtion for the mixed A.O.V. model using the W transformation, Technometrics 15, 819-831.

Henderson, C.R., Kempthorne, O., Searle, S.R., and Von Krosigk, C.N. (1959), Estimation of environmental and genetic trends form records subject to culling, Biometrics, 15, 192-218.

Henderson, H.V., and Searle, S.R. (1981), On deriving the inverse of a sum of matrices, SIAM Review, 23, 53-60.

Hovland, P., Bischof, C., Spiegelman, D., and Casella, M. (1997), Efficient derivative codes through automatic differentiation and interface contraction: an application in biostatistics, SIAM Journal on Scientific Computing, 18, 1056-1066.

Jennrich, I.R., and Schluchter, M.D. (`1986), Unbalanced repeated-measure models with structured covariance matrices, Biometrics, 42, 805-820.

Jensen, J., and Mao, I.L. (1988), Transformation algorithms in analysis of single trait and multitrait models with equal design matrices and one random factor per trait: a review, Journal of Animal Science, 66, 2750-2761.

Johnson, D.L., and Thompson, R. (1995), Restricted maximum likelihood estimation of variance components for univariate animal models using sparse matrix techniques and average information, Journal of Dairy Science, 78, 449-456.

Laird, N.M. (1982), Computing variance components using the EM algorithm, Journal of

Statistical Computation and Simulation, 14, 295-303.

Laird, N.M., Lange, N., and Stram, D. (1987), Maximum likelihood computation with repeated measures: application of the EM algorithm, Journal of the American Statistical Association, 82, 97-105.

Lindstrom, J.M., and Bates, D.M. (1988), Newton-Raphson and the EM algorithms for linear mixed-effects models for repeated measures data, Journal of the American Statistical Association, 83, 1014-1022.

Meyer, K. (1985), Maximum likelihood estimation of variance components for a multivariate mixed model with equal design matrices, Biometrics, 41, 153-166.

Meyer, K. (1989), Restricted maximum likelihood to estimate variance components with several random effects using a derivative-free algorithm, Génétique Sélection and Evolution, 21, 317-340.

Meyer, K. (1991), Estimating variances and covariances for multivariate animal models by restricted maximum likelihood, Génétique Sélection and Evolution, 23, 67-83.

Meyer, K. (1997), An “average information” restricted maximum likelihood algorithm for estimating reduced rank genetic covariance matrices or covariance functions for animal models with equal design matrices, Génétique Sélection and Evolution, 29, 97-116.

Meyer, K., and Smith, S.P. (1996), Restricted maximum likelihood estimation for animal models using derivatives of the likelihood, Génétique Sélection and Evolution, 28, 23-49.

Misztal, I. (1990), Restricted maximum likelihood estimation of variance components in animal models using sparse matrix inversion and a supercomputer, Journal of Dairy Science, 73 (1990), pp. 163-172.

Misztal, I., and Perez-Enciso, M. (1993), Sparse matrix inversion for restricted maximum likelihood estiamtion of variance components by expectation-maximization, Journal of Dairy Science 76, 1479-1483.

Mohammadi, B., Male, J.-M., Rostaing-Schmidt, N. (1996), Automatic Differentiation in Direct and reverse modes: application to optimum shape design in fluid mechanics, in Computational differentiation: Techniques, applications, and tools, eds. M. Berz, C. Bischof, G. Corliss, and A. Griewank , Philadelphia: SIAM, pp. 309-318.

Neumaier, A., and Groeneveld, E. (1998), Restricted maximum likelihood estimation of covariances in sparse linear models, Génétique Sélection and Evolution, 30, 3-26.

Patterson, H.D., and Thompson, R. (1971), Recovery of inter-block information when block sizes are unequal, Biometrika, 58, 545-554.

Searle, S.R. (1989), Variance components-some history and a summary account of estimation methods, Journal of Animal Breeding and Genetics, 106, 1-29.

Smith, S.P. (1995), Differentiation of the Cholesky Algorithm, Journal of Computational and Graphical Statistics, 4, 134-147.

Smith, S.P., and Graser, H.-U. (1986), Estimating variance components in a class of mixed models by restricted maximum likelihood, Journal of Dairy Science, 69, 1156-1165.

Smith, S.P., and Hammond, K. (1988), Rank regression with log gamma residuals. Biometrika, 75, 741-751.

Takahashi, K., Fagan, J., and Chen, M. (1973), Formation of a sparse bus impedance matrix and its application to short circuit study, in 8th Power Industry Computer Applications Conference, New York: IEEE, pp. 63-69.

Tier, B., and Smith, S.P. (1989), Use of sparse matrix absorption in animal breeding, Génétique Sélection and Evolution, 21, 457-466.

Wolfinger, R., Tobias, R., and Sall, J. (1994), Computing Gaussian likelihoods and their derivatives for general linear mixed models, SIAM Journal on Scientific Computing, 15, 1294-1310.