The Backward Differentiation of the Bordering Algorithm for an Indefinite Cholesky Factorization

August 2017
Stephen P. Smith
email: hucklebird@aol.com

Abstract

The bordering method of the Cholesky decomposition is backward differentiated to derive a method of calculating first derivatives. The result is backward differentiated again and an algorithm for calculating second derivatives results. Applying backward differentiation twice also generates an algorithm for conducting forward differentiation. The differentiation methods utilize three main modules: a generalization of forward substitution for calculating the forward derivatives; a generalization of backward substitution for calculating the backward derivatives; and an additional module involved with the calculation of second derivatives. Separating the methods into three modules lends itself to optimization where software can be developed for special cases that are suitable for sparse matrix manipulation, vector processing and/or blocking strategies that utilize matrix partitions. Surprisingly, the same derivative algorithms fashioned for the Cholesky decomposition of a positive definite matrix can be used again for matrices that are indefinite. The only differences are very minor adjustments involving an initialization step that leads into backward differentiation and a finalization step that follows forward differentiation.

1. Introduction

General methods to conduct forward and backward differentiation are well described by Griewank (1989, 2000). Smith (1995a) described the forward and backward derivatives that were particular to the Cholesky decomposition computed by the outer-product form. Murray (2016) provided those same derivatives for the Cholesky decomposition computed by the inner-product form. Smith (2017) provided the backward differentiation for the Cholesky decomposition computed by the bordering method. Moreover, there are now available symbolically derived derivatives for the Cholesky decomposition (and the LU factorization) that are algorithm independent (De Hoog, Anderssen and Lukas, 2011, Koerber 2015, Murray 2016). What went unanticipated in 1995 was how open-ended and varied the different approaches were to become. The three standard ways to compute the Cholesky decomposition are all distinguished by a re-ordering of the calculations. When algorithmic differentiation is applied over these different variants of the Cholesky decomposition, what returns is a new algorithm that comes with unique

properties that might find special advantages with particular applications. These different approaches can all be explored, in a search for a better algorithm that might be superior for sparse-matrix manipulation, or vector processing or blocking strategies.

The purpose of the present paper is to expand on the methods that emerge from the bordering algorithm of the Cholesky decomposition, an effort initiated by Smith (2017) that provided only first derivatives. The present paper will provide the second derivatives, but also generalize the Cholesky decomposition for treating indefinite matrices. Some theoretical considerations for factorizing an indefinite matrix are provided by Smith(2001a). In application, indefinite equations emerge naturally with Siegel's (1965) method to treat least squares. Smith (2001b) described restricted maximum likelihood using the Cholesky decomposition of an indefinite matrix, and also provided first and second derivatives that came with the algorithmic differentiation of the Cholesky's factorization of an indefinite matrix using the outer-product form. Smith, Nikolic and Smith (2012) applied Cholesky's factorization of an indefinite matrix to a problem in particle physics, and also posted corrections to the differentiation method presented by Smith (2001b).

The indefinite Cholesky decomposition is represented by the following.

(1) $\mathbf{L}\Delta\mathbf{L}^{\mathrm{T}}=\mathbf{M}$

Where **L** is a lower triangular matrix and Δ is a diagonal matrix with diagonals 1 or -1. The matrix **M** is the symmetric and indefinite matrix that is subject to factorization. One of the remarkable discoveries reported in the present paper is that differentiation can proceed by using Δ in initialization, and then continuing with algorithms developed for the case when **M** is more normally positive definite. In other words, treating the indefinite case is a mere tack-on of Δ to the methods developed for the positive definite case. This observation was missed in Smith (2001b), but it is apparent with the symbolic differentiation that resembles forward differentiation. To see this define the shorthand matrices $\mathbf{L}_x$ and $\mathbf{M}_x$ below.

$$\mathbf{L}_x = \frac{\partial \mathbf{L}}{\partial x}$$

$$\mathbf{M}_x = \frac{\partial \mathbf{M}}{\partial x}$$

Differentiating both side of (1) gives the following.

(2) $\mathbf{L}_x\,\Delta\mathbf{L}^{\mathrm{T}} + \mathbf{L}\Delta\mathbf{L}_x^{\mathrm{T}} = \mathbf{M}_x$

Pre-multiplying equation (2) by $\mathbf{L}^{-1}$ and post-multiplying by $\mathbf{L}^{-1\,\mathrm{T}}$ (or $\mathbf{L}^{-\mathrm{T}}$ for short), gives the following where the left-hand side is recognized as the sum of a lower triangular

matrix and an upper triangular matrix.

$$\mathbf{L}^{-1}\mathbf{L}_x \Delta + \Delta \mathbf{L}_x^T \mathbf{L}^{-T} = \mathbf{L}^{-1}\mathbf{M}_x \mathbf{L}^{-T}$$

As others[1] have done, define the matrix function $\Phi(\mathbf{A})$=Lower($\mathbf{A}$)-½Diag($\mathbf{A}$) where Lower($\mathbf{A}$) is the lower triangular part of $\mathbf{A}$ and Diag($\mathbf{A}$) is diagonal with the diagonals of $\mathbf{A}$. Then it becomes apparent that $\mathbf{L}^{-1}\mathbf{L}_x \Delta = \Phi(\mathbf{L}^{-1}\mathbf{M}_x \mathbf{L}^{-T})$, and solving for $\mathbf{L}_x$ gives the sought derivatives:

$$\mathbf{L}_x = \mathbf{L}\, \Phi(\mathbf{L}^{-1}\mathbf{M}_x \mathbf{L}^{-T}) \Delta$$

In other words, the same calculations are used that assume $\mathbf{M}$ is positive definite, and then the result is post-multiplied by Δ in the most trivial of post-hoc adjustments.

The present paper takes the permutation order of $\mathbf{M}$ that makes it factorable, as provided. A symbolic factorization is possible when $\mathbf{M}$ is positive definite. However, when $\mathbf{M}$ is indefinite the permutation order is found dynamically using real number calculations. This is accomplished with the outer-product form, and it is very expensive[2] for large and sparse matrices where a double linked-list is used. Because $\mathbf{M}$ is assumed to be a continuous function of parameters and is subject to differentiation, in theory a different permutation order may be needed for each parameter set. This requirement would defeat the present paper where attention is given to the bordering method rather than the outer product form. Fortunately, once a permutation order is found, and along with the sparse structure, it is unlikely that a different permutation order is needed for alternative selections of the parameter set. Neither does Δ change for different selections of the parameter set.

In Section 2, the bordering method is described for the indefinite Cholesky decomposition. All the algorithms described in this paper will be presented in modular form as much as possible. This ads a layer of complexity, but the return for doing this is great given that each module can be optimized independently. In Sections 3 and 4, forward substitution, backward substitution and their generalizations are presented. These operations will become primal modules and used for the first and second derivative calculations presented in Sections 5 and 6. It is found that backward differentiation uses a generalized backward substitution and forward differentiation uses a generalized forward substitution.

Regarding notation, matrices and vectors will be represented in bold, as already illustrated. When a lower triangular matrix or symmetric matrix $\mathbf{A}$ is indicated as half-stored, the ij-th element given in the i-th row and j-th column of $\mathbf{A}$ is denoted by A_{ij},

---

[1] De Hoog, Anderssen and Lukas (2011), and Murray (2016), in particular.

[2] In terms of computing time and memory.

where $j \le i$. Occasionally, the algorithms will automatically select an element A_{ij} where $j>i$ and in this case it is always understood that $A_{ij}=A_{ji}$. Likewise, the full-stored version of a lower triangular matrix **L** is defined to be $\mathbf{L}+\mathbf{L}^T$-Diag(**L**).

2. Bordering Algorithm for an Indefinite Matrix

Notation required to define the bordering method is presented below.

$$\mathbf{A}_{k+1}=\begin{bmatrix}\mathbf{A}_k & \mathbf{a}_{k+1}\\ \mathbf{a}_{k+1}^T & \alpha_{k+1}\end{bmatrix}=\begin{bmatrix}\mathbf{L}_k & \\ \mathbf{u}_{k+1}^T & d_{k+1}\end{bmatrix}\begin{bmatrix}\Delta_k & \\ & \delta_{k+1}\end{bmatrix}\begin{bmatrix}\mathbf{L}_k^T & \mathbf{u}_{k+1}\\ & d_{k+1}\end{bmatrix}=\mathbf{L}_{k+1}\Delta_{k+1}\mathbf{L}_{k+1}^T$$

where:

$$\mathbf{A}_k=\mathbf{L}_k\Delta_k\mathbf{L}_k^T$$
$$\mathbf{u}_{k+1}=\Delta_k\mathbf{L}_k^{-1}\mathbf{a}_{k+1}$$
$$d_{k+1}=\sqrt{\delta_{k+1}\left(\alpha_{k+1}-\mathbf{u}_{k+1}^T\Delta_k\mathbf{u}_{k+1}\right)}$$

and Δ_k is a diagonal matrix with diagonals 1 or -1, δ_k is a scalar being 1 or -1, all depending on whether the diagonals or scalar represent the positive or negative partition of the N×N matrix $\mathbf{A}_N=\mathbf{M}$. Both Δ_k and δ_k can be represented implicitly from the initial matrix structure of $\mathbf{A}_N$, and never change. Note than $\Delta_k^{-1}=\Delta_k$.

The bordering method to compute the Cholesky decomposition of the indefinite matrix **M**, with Δ_N provided, is given below.

1. Set $\mathbf{A}_N=\mathbf{M}$, thereby defining all the arrays, $\mathbf{A}_k$, $\mathbf{a}_k$, α_k, $N \ge k>1$, and $A_1=\alpha_1$, implicitly as data entries.

2. Evaluate $d_1=(\delta_1\,\alpha_1)^{1/2}$.

3. For k=1, 2, ... N-1, perform the following calculations.

   A) Solve **v** in the lower triangular system, $\mathbf{L}_k\mathbf{v}=\mathbf{a}_{k+1}$, by forward substitution, then set $\mathbf{u}_{k+1}=\Delta_k\mathbf{v}$.

   B) Evaluate the vector product, $\xi=\mathbf{u}_{k+1}^T\Delta_k\mathbf{u}_{k+1}$, and then evaluate $d_{k+1}=[\delta_{k+1}(\alpha_{k+1}-\xi)]^{1/2}$.

The matrix **M** can be half stored, and because its entries are used only once in one of the above calculations, it is feasible to overwrite **M** with $\mathbf{L}_N$ while following the bordering

method.

3. Generalized Forward Substitution

It is useful to generalize matrix routines to permit streamlining thereby improving numerical efficiency and to permit modulation thereby aiding the chore of programming.

Forward substitution to solve the lower triangular system, **Lv=a**, is well known and indicated in Display 1. Initialize the function call by providing **L** (lower triangular) with L_{ij}, $j \le i$, and set $\mathbf{v} \leftarrow \mathbf{a}$ with call to FS(**L**,**a**,N).

Display 1. Forward Substitution, denoted by FS(**L**,**v**,N).

$$
\begin{aligned}
&For\ i = 1\ to\ N \\
&\quad v_i \leftarrow \frac{v_i}{L_{ii}} \\
&\quad For\ j = i+1\ to\ N \\
&\qquad v_j \leftarrow v_j - v_i \times L_{ji} \\
&\quad end\ j; \\
&end\ i
\end{aligned}
$$

The routine of Display 1 overwrites the vector **a** with the solution **v**, and hence only one vector **v** is needed in the routine.

Denote the above forward substitution algorithm by FS(**L**,**v**,N). Make one modification to FS(**L**,**v**,N) to generate a new algorithm that can be suspended once index-i becomes $k \le N$. The new procedure is presented in Display 2, and is denoted by GFS(**L**,**v**,k,N).

Display 2. Generalized Forward Substitution, denoted by GFS(**L**,**v**,k,N).

$$
\begin{aligned}
&For\ i = 1\ to\ k \\
&\quad v_i \leftarrow \frac{v_i}{L_{ii}} \\
&\quad For\ j = i+1\ to\ N \\
&\qquad v_j \leftarrow v_j - v_i \times L_{ji} \\
&\quad end\ j; \\
&end\ i
\end{aligned}
$$

Make an additional modification to GFS(**L**,**v**,k,N) by postponing[3] all the subtractions until when they are needed prior to the respective division. The new algorithm, denoted by GFS1(**L**,**v**,k), is presented in Display 3. GFS1(**L**,**v**,k) provides an alterative calculation to GFS(**L**,**v**,k,N) only when k=N.

Display 3. Generalized Forward Substitution, version 1 denoted by GFS1(**L**,**v**,k).

$$
\begin{aligned}
&For\ i = 1\ to\ k \\
&\quad For\ j = 1\ to\ i-1 \\
&\qquad v_i \leftarrow v_i - v_j \times L_{ij} \\
&\quad end\ j \\
&\quad v_i \leftarrow \frac{v_i}{L_{ii}} \\
&end\ i
\end{aligned}
$$

[3]Operations are postponed not to simplify mathematical derivations. That this is done inflates the number of possible algorithms that are derived from one simple algorithm. Rather, operations are postponed because with sparse-matrix manipulation (even vector processing) it is more efficient to treat all the subtractions together in appropriately defined groups.

For illustration purposes, make three additional changes to GFS1(**L**,**v**,k), turning it into GFS2(**L**,Δ,k): implicitly make **v** part of **L**, where $v_i=L_{ik}$; and suspend the last division where the index-i becomes k, but additionally intervene before the index-i becomes k and multiply L_{ki} and its square by δ_i or the i-th diagonal of Δ[4]. These changes make the following algorithm.

Display 4. Generalized Forward Substitution, version 2 denoted by GFS2(**L**,Δ,k).

$$
\begin{aligned}
&For\ i = 1\ to\ k-1\\
&\quad For\ j = 1\ to\ i-1\\
&\qquad L_{ki} \leftarrow L_{ki} - L_{kj} \times L_{ij}\\
&\quad end\ j\\
&L_{ki} \leftarrow \frac{L_{ki}}{L_{ii}}\\
&end\ i\\
&For\ j = 1\ to\ k-1\\
&\quad L_{kj} \leftarrow \delta_j \times L_{kj}\\
&\quad L_{kk} \leftarrow L_{kk} - \delta_j \times L_{kj} \times L_{kj}\\
&end\ j\\
&L_{kk} \leftarrow \delta_k L_{kk}
\end{aligned}
$$

With these routines defined, shorthand representations of the indefinite Cholesky decomposition and its derivatives become available. In particular, the bordering algorithm presented in Section 2 becomes the following.

Δ *provided*
L←*half-stored(**M**)*

For k=1 to N
Call the routine GFS2(**L**,Δ,k)
$L_{kk} = \sqrt{L_{kk}}$

[4]The multiplication by δ_i is done to conform with the bordering algorithm in Section 2, as will become apparent shortly. Note that $\delta_i = \delta_i \times \delta_i \times \delta_i$.

4. Generalized Backward Substitution

Backward substitution to solve the upper triangular system, $\mathbf{L}^T\mathbf{v}=\mathbf{a}$, is well known and indicated in Display 5. To initialize, provide $\mathbf{L}^T$ upper triangular with L_{ij}, $j \le i$, and set $\mathbf{v} \leftarrow \mathbf{a}$ by the function call BS(**L**,**a**,N) .

Display 5. Backward substitution algorithm denoted by BS(**L**,**v**,N).

$$
\begin{aligned}
&For\ i = N\ to\ 1 \\
&\quad v_i \leftarrow \frac{v_i}{L_{ii}} \\
&\quad For\ j = i-1\ to\ 1 \\
&\qquad v_j \leftarrow v_j - v_i \times L_{ij} \\
&\quad end\ j; \\
&end\ i
\end{aligned}
$$

As was accomplished for forward substitution, make two modifications to BS(**L**,**v**,N): postpone all the subtractions until when they are needed prior to the respective division; make it so the routine can start with the index-i beginning with k rather than N. The new algorithm, denoted by GBS(**L**,**v**,k,N), is presented in Display 6.

Display 6. Generalized backward substitution algorithm denoted by GBS(**L**,**v**,k,N).

$$
\begin{aligned}
&For\ i = k\ to\ 1 \\
&\quad For\ j = i+1\ to\ N \\
&\qquad v_i \leftarrow v_i - v_j \times L_{ji} \\
&\quad end\ j \\
&\quad v_i \leftarrow \frac{v_i}{L_{ii}} \\
&end\ i
\end{aligned}
$$

5. First Derivatives by Backward Differentiation

The bordering method is highly granular because all the main intermediate calculations are saved and are not lost to overwriting. Except for the possibility of overwriting various parts of the initial data, or the matrix **M**, with various parts of $\mathbf{L}_N$ as they are computed, the application of the *Rules for Backward Differentiation*[5] are directly applicable. Even when matrix **M** is overwritten, enough information is saved in $\mathbf{L}_N$ to propagate the derivatives backward to the initial data with little difficulty. Therefore, it is advantages to permit what little overwriting that may exists. If **F** is the array that stores the backward propagated derivatives (following the Rules for Backward Differentiation) then let $\mathbf{F}_{L(N)}$ correspond to all the non-data intermediates that are all neatly collected in $\mathbf{L}_N$ and let $\mathbf{F}_M$ correspond to the intermediates given by the initial data, or **M**, that is overwritten in the forward sweep. The overwriting is represented by $\mathbf{F}_{L(N)} \leftarrow \mathbf{F}_M$ in the backward sweep where $\mathbf{F}_M$ signifies a half-stored matrix. The symbolic representations preserve the distinction between $\mathbf{F}_{L(N)}$ and $\mathbf{F}_M$, even though $\mathbf{F}_{L(N)}$ is lost by overwriting, and that is enough for our purpose. Likewise, let $\mathbf{F}_{L(k)}$, $\mathbf{F}_{u(k)}$, and $F_{d(k)}$ correspond to the intermediates of $\mathbf{L}_k$, $\mathbf{u}_k$ and d_k, all belonging to the larger array $\mathbf{F}_{L(N)}$. Let $\mathbf{F}_{A(k)}$, $\mathbf{F}_{a(k)}$ and $F_{\alpha(k)}$ correspond to $\mathbf{A}_k$, $\mathbf{a}_k$, α_k, all belonging to $\mathbf{F}_M$. More generally, let F_{ij} correspond to the intermediate represented by L_{ij} or M_{ij}.

The Rules for Backward Differentiation can be applied directly to the algorithm in Section 2, where a slight modification is needed for Step 3A that indicates a non-scalar function. That first approach was followed by Smith (2017), but for the case that $\Delta=\mathbf{I}$. Alternatively, the rules can be applied directly to the algorithm presented in Section 3 (i.e., the nested application of GFS2), with no modification. That algorithm is listed in Display 7 again, but with embellishment to aid differentiation by hand.

---

[5]Rules for Backward Differentiation follow form Griewank (1989). They are provided in memos by Smith (1995b, pg 13; and 2000, Section 4.2), as symbolic tools that can be used directly by a programer. In the forward sweep the k-th recursion is $h_k=f_k(\Omega_k)$, for $\Omega_k \subseteq \{h_i : i<k\}$, k=1, 2, ...r; then in the reverse sweep, backward derivatives are accumulated by, $F_{h(i)} = F_{h(i)} + F_{h(k)} \times \partial f_k/\partial h_i$, for all $h_i \in \Omega_k$, k=r, ... 2, 1, such that F is an array corresponding to all the intermediates where F_h represents h(), and F is suitably initialized to F= null except for $F_{h(r)} = 1$ where h_r is a scalar. If a step also involves overwriting, where the k-th recursion is typically of the form $h_i \leftarrow f_k(\Omega_k)$, i.e., where h_k overwrites an element of Ω_k as $h_i \leftarrow h_k$, then the update is $F_{h(i)} \leftarrow F_{h(k)} \times \partial f_k/\partial h_i$ for the particular $h_i \in \Omega_k$ that changes status. Overwriting is vary problematic when some of the partial derivatives, $\partial f_k/\partial h_j$ for $h_j \in \Omega_k$, are unavailable because some of the elements of Ω_k are lost due to overwriting.

Display 7. Bordering method with embellishments to aid differentiation by hand.

$$
\begin{aligned}
&For\ k = 1\ to\ N\\
&For\ i = 1\ to\ k-1\\
&\quad For\ j = 1\ to\ i-1\\
&\qquad L_{ki} \leftarrow \overline{L}_{ki} - \hat{L}_{kj} \times L_{ij}\\
&\quad end\ j\\
&L_{ki} \leftarrow \frac{\hat{L}_{ki}}{L_{ii}}\\
&end\ i\\
&For\ j = 1\ to\ k-1\\
&\quad L_{kj} \leftarrow \delta_j \times \hat{L}_{kj}\\
&\quad L_{kk} \leftarrow \overline{L}_{kk} - \delta_j \times L_{kj} \times L_{kj}\\
&end\ j\\
&L_{kk} \leftarrow \sqrt{\delta_k \hat{L}_{kk}}\\
&end\ k
\end{aligned}
$$

Intermediates of Display 7 to the right of the arrows fall into three classifications. If it has no hat nor an over-bar, the intermediate is not overwritten by the algorithm. Those intermediates are available during the reverse sweep. If the intermediate has an over-bar then it is lost due to overwriting, but in a way that brings no complexity to the Rules of Backward Differentiation. If the intermediate has a hat, then it is lost with overwriting and special adjustments and precautions are needed with hand differentiation: namely, when the needed partial derivatives are sought, they must be imputed from the information that is available and fitted together in a way that is consistent with the intended calculous. Without these distinctions it is easy to muddle the process and produce incorrect results.[6] Otherwise, a correct application of the reverse sweep, starting at the bottom and working through the list backwards, produces the algorithm presented in Display 8. Note that the correct application has various δ_j permeating through it.

[6]This provides a justification for using an automatic tool, even one that provides the symbolic manipulation done here. However, humans deserve employment too.

Display 8. Symbolic backward derivatives of the bordering method.

1. Initialize $\mathbf{F}_L$ to the firth derivatives of f(**L**) where f() is a scalar function of **L**, as indicated below.

$$\mathbf{F}_L \leftarrow \frac{\partial f(\mathbf{L})}{\partial \mathbf{L}}$$

2. For k=N to 1 perform the following recursive calculations.

$$F_{kkc} \leftarrow \frac{F_{kkc}}{2 \times L_{kkc}}$$
$$F_{kkc} \leftarrow \delta_k F_{kkc}$$
$$For\ j = k-1\ to\ 1$$
$$\quad F_{kj} \leftarrow F_{kj} - 2\delta_j L_{kj} F_{kkc}$$
$$\quad F_{kj} \leftarrow \delta_j F_{kj}$$
$$end\ j$$
$$For\ i = k-1\ to\ 1$$
$$\quad F_{ki} \leftarrow \frac{F_{ki}}{L_{ii}}$$
$$\quad F_{ii} \leftarrow F_{ii} - \delta_i L_{ki} F_{ki} \qquad **$$
$$\quad For\ j = i-1\ to\ 1$$
$$\quad\quad F_{kj} \leftarrow F_{kj} - L_{ij} F_{ki}$$
$$\quad\quad F_{ij} \leftarrow F_{ij} - \delta_j L_{kj} F_{ki} \qquad **$$
$$\quad end\ j$$
$$end\ i$$

The calculation in Step 2 of Display 8 can be rearranged. Firstly, the division of F_{kk} by 2 can be withheld to the end. Secondly, the multiplications involving Δ, namely $\delta_k F_{kk}$ and $\delta j F_{kj}$, can be first up.[7] Lastly, the second summation involving index-j can be combined

[7]To see this note that $\delta_j=\delta_j\times\delta_j$ and $F_{kj}\leftarrow F_{kj}-2\delta_j L_{kj} F_{kk}$ implies $\delta_j F_{kj}\leftarrow\delta_j F_{kj}-2L_{kj} F_{kk}$.

with the first by removing the starred operations and letting index-i range from k to 1. The starred operations can be separated and moved to their own summation. These changes produce the interesting recursions for Step 2 that are presented in Display 9.

Display 9. Rearranged derivative calculations.

2. For k=N to 1 perform the following recursive calculations.

$$F_{kj} \leftarrow \delta_j F_{kj}, \; j = k \; to \; 1$$
$$For \; i = k \; to \; 1$$
$$F_{ki} \leftarrow \frac{F_{ki}}{L_{ii}}$$
$$F_{kj} = F_{kj} - L_{ij} F_{ki}, \; j = i-1 \; to \; 1$$
$$end \; i$$
$$For \; i = k-1 \; to \; 1$$
$$F_{ij} \leftarrow F_{ij} - \delta_j L_{kj} F_{ki}, \; j = i \; to \; 1 \quad **$$
$$end \; i$$
$$F_{kk} \leftarrow \frac{F_{kk}}{2}$$

The starred operations make adjustments to $\mathbf{F}_L$ that follow calculating F_{ki}, $i \leq k$, in the k-th step. These can all be postponed until when they are needed. A surprising[8] observation is that the multiplication of δ_j for the starred calculations cancels with the up-front multiplications that are waiting their turn to initialize backward substitution when k=i and leading to $\delta_j \times \delta_j = 1$. This lets the entire Δ matrix be factored out of the algorithm and applied with the initialization step for $\mathbf{F}_L$. Excluding the first multiplications involving Δ, and the starred operations, and the last division by 2, what is left is recognized as simple backward substitution. Moreover, the starred or postponed operations (with δ_j dropped) are precisely those adjustments that are made with a generalized backward substitution. This revelation agrees exactly with Smith (2017) but now generalized to include Δ. More

[8]This is less surprising once the symbolic differentiation presented in Section 1 is appreciated, and noting that the difference between forward and backward differentiation can be described as the order matrices (each representing partial differentiation) are multiplied together (Greiwank 1992). The result is surprising, nevertheless, because Δ remains hopelessly entangled in the bordering method.

importantly, with Δ factored completely out then any suitable algorithm that backward differentiates the Cholesky decomposition (for a positive definite matrix) can be used carte blanche, but our focus is with the bordering method.

These remarkable results are now rewritten to express backward differentiation in modular form. First define the vector $\mathbf{v}_k$ as the k-th column of the full-stored version of $\mathbf{F}_L$, namely $\mathbf{F}_L + \mathbf{F}_L^T - \text{Diag}\{\mathbf{F}_L\}$. Partition $\mathbf{v}_k$ into $\mathbf{v}_k^T = (\underline{\mathbf{v}}_k^T, \overline{\mathbf{v}}_k^T)$, where $\underline{\mathbf{v}}_k$ are the elements of $\mathbf{F}_L^T$ above and including the k-th diagonal and $\overline{\mathbf{v}}_k^T$ are the elements of $\mathbf{F}_L$ strictly below the k-th diagonal. The vector $\underline{\mathbf{v}}_k$ represents elements of $\mathbf{F}_L$ that are being fully computed during the k-th step, whereas the vector $\overline{\mathbf{v}}_k$ corresponds to elements of $\mathbf{F}_L$ that are already computed but required for the k-th step. Backward differentiation is neatly provided by the following algorithm.

1. Initialize $\mathbf{F}_L$ to the firth derivatives of f($\mathbf{L}$) where f() is a scalar function of $\mathbf{L}$ and multiply by Δ, as indicated below.

$$\mathbf{F}_L \leftarrow \frac{\partial f(\mathbf{L})}{\partial \mathbf{L}} \times \Delta$$

2. Denote this step by BD($\mathbf{L}$,$\mathbf{F}_L$,N). For k=N to 1 perform the following calculations:

A) Extract $\mathbf{v}_k$ as the k-th column of $\mathbf{F}_L + \mathbf{F}_L^T - \text{Diag}\{\mathbf{F}_L\}$.
B) Call GBS($\mathbf{L}$,$\mathbf{v}_k$,k,N)
C) Multiply the k-th element of $\mathbf{v}_k$ by ½
D) Write $\underline{\mathbf{v}}_k$ to storage device holding $\mathbf{F}_L$

There are some significant advantages to this algorithm. First, for treating large sparse matrices, the bordering algorithm for computing $\mathbf{L}$, and the associated generalized backward substitution, can be streamlined to avoid intense searching of the non-zero entries before performing the sought non-trivial calculations. Rather, the non-zero elements can be found together and ready for fast calculation, provided a preliminary factorization was performed to define the sparse structure and including needed row-column permutations. Second, even when $\mathbf{L}$ (or $\mathbf{M}$) is a large matrix, only one column of $\mathbf{F}_L + \mathbf{F}_L^T - \text{Diag}\{\mathbf{F}_L\}$ need be available during step-k with $\mathbf{L}$ completely stored in memory, letting the entire $\mathbf{F}_L$ be computed with no additional random-access memory.

With $\mathbf{F}_L$ turned $\mathbf{F}_M$ computed when BD($\mathbf{L}$,$\mathbf{F}_L$,N) returns, all first derivatives are evaluated by the following.

$$\frac{\partial f(\mathbf{L})}{\partial x} = \sum_{ij} F_{ij} \frac{\partial M_{ij}}{\partial x}$$

6. Second Derivatives by Backward Differentiation

Following the tradition of Smith (1995a), two rounds of backward differentiation is presented below, rather than an alternative combination of forward and backward differentiation. To accomplish this, the algorithms presented in Sections 3 and 5 are strung out in outline form from start to finish, and the backward sweep is then revealed by proceeding with the Rules for Backward Differentiation from the end of the list back to the beginning. Some definitions are needed. Firstly, the work vector $\mathbf{v}_k$ used in BD($\mathbf{L}$,$\mathbf{F}_L$,N) is to be replaced with a direct reference to $\mathbf{F}$: i.e., the i-th element of $\mathbf{v}_K$ is assigned to F_{ik} (if $i \geq k$) or F_{ki} (if $i<k$). It is also necessary to introduce two new arrays, $\mathbf{S}$ and $\mathbf{Q}$, to keep track of intermediate calculation within GBS($\mathbf{L}$,$\mathbf{v}_k$,k,N) that involve $\mathbf{L}$ and $\mathbf{F}$, respectively: make the following correspondences, $S_{ij} \leftrightarrow L_{ij}$ and $Q_{ij} \leftrightarrow F_{ij}$.

The outline of factorization and one round of backward differentiation is presented in Display 10.

Display 10. Recursion list for the bordering method and its backward derivatives.

$$1.\ \Delta\ provided, \mathbf{L} \leftarrow half - stored(\mathbf{M})$$
$$For\ k = 1\ to\ N$$
$$Call\ GFS2(\mathbf{L}, \Delta, k)$$
$$L_{kk} \leftarrow \sqrt{L_{kk}}$$
$$2.\ \mathbf{F} \leftarrow \frac{\partial f(\mathbf{L})}{\partial \mathbf{L}} \times \Delta$$
$$3.\ For\ k = N\ to\ 1$$
$$For\ i = k\ to\ 1$$
$$For\ m = i + 1\ to\ N$$
$$F_{ki} \leftarrow F_{ki} - F_{mk} L_{mi} \quad (note: F_{mk} = F_{km},\ when\ k > m)$$
$$end\ m$$
$$F_{ki} \leftarrow \frac{F_{ki}}{L_{ii}}$$
$$end\ i$$
$$4.\ F_{ii} \leftarrow \frac{\hat{F}_{ii}}{2}, i = 1\ to\ N \qquad (note: \hat{F}_{ii}\ lost\ to\ overwriting)$$
$$5.\ \frac{\partial f(\mathbf{L})}{\partial x} = \sum_{ij} F_{ij} \frac{\partial M_{ij}}{\partial x}$$

By going through the above list of Display 10 in reverse, and applying the Rule of Backward Differentiation, the recursive operations presented in Display 11 are generated. These calculations depend on the elements of **S** being initialized to zero.

Display 11. Symbolic backward derivatives of the Display 10 recursions.

$$5.\ \mathbf{Q} \leftarrow \frac{\partial \mathbf{M}}{\partial x}$$

$$4.\ Q_{ii} \leftarrow \frac{Q_{ii}}{2},\ i = 1\ to\ N$$

$$3.\ For\ k = 1\ to\ N$$

$$For\ i = 1\ to\ k$$

$$Q_{ki} \leftarrow \frac{Q_{ki}}{L_{ii}}$$

$$S_{ii} \leftarrow S_{ii} - Q_{ki}\hat{F}_{ki} \quad (note\ \hat{F}_{ki} = F_{ki}\ if\ k \neq i\ or\ \hat{F}_{kk} = 2 \times F_{kk})$$

$$For\ m = i + 1\ to\ N$$

$$Q_{mk} \leftarrow Q_{mk} - Q_{ki}L_{mi} \quad (note: Q_{mk} = Q_{km}\ when\ k > m)$$

$$S_{mi} \leftarrow S_{mi} - Q_{ki}\hat{F}_{mk} \quad (note: \hat{F}_{mk} = \hat{F}_{km}\ when\ k > m)$$

$$end\ m$$

$$end\ i$$

$$2.\ \mathbf{S} \leftarrow \mathbf{S} + \sum_{i \geq j} \delta_j Q_{ij} \frac{\partial^2 f(\mathbf{L})}{\partial \mathbf{L} \partial L_{ij}}$$

$$1.\ \mathbf{S} \leftarrow \mathbf{S} \times \Delta\ then\ call\ BD(\mathbf{L}, \mathbf{S}, N)$$

$$0.\ \frac{\partial^2 f(L)}{\partial x \partial y} = \sum_{ij} F_{ij} \frac{\partial^2 M_{ij}}{\partial x \partial y} + \sum_{ij} S_{ij} \frac{\partial M_{ij}}{\partial y}$$

Some important observations can now be made with the Display 11 protocol for calculating second derivatives. Firstly, the impact of Δ is limited to Step 2, and this adjustment is remarkably trivial and mirrors what was found for the first derivative calculation. Secondly, the array **Q** can be calculated separately in Step 3, and then with **Q** already calculated the array **S** can be computed. Thirdly, if it where not for the factored out matrix Δ, the calculation of **Q** is the forward differentiation calculation (Smith 2000). More importantly, the calculation of **Q** is a nested application of the GFS(**L**,$\mathbf{v}_k$,k,N) algorithm, where k varies between 1 and N and where $\mathbf{v}_k$ is the k-th column of

Q+**Q**^{T}-Diag(**Q**) in the process of computation. All the above can now be rewritten is a form that's ready for application, and presented in Display 12.

Display 12. Protocol for calculating second derivatives.

$$5.\ \mathbf{Q} \leftarrow \frac{\partial \mathbf{M}}{\partial x}$$

$$4.\ Q_{ii} \leftarrow \frac{Q_{ii}}{2},\ i = 1\ to\ N$$

$$3.A)\ For\ k = 1\ to\ N$$

$$Set\ \mathbf{q}_k = k-th\ column\ of\ \mathbf{Q} + \mathbf{Q}^T - Diag\{\mathbf{Q}\}$$

$$Call\ GFS(\mathbf{L}, \mathbf{q}_k, k, N)$$

$$Output\ \mathbf{q}_k\ to\ \mathbf{Q}\ storage$$

$$3.B)\ Initialize\ S_{ij} = 0,\ all\ j \le i$$

$$For\ k = 1\ to\ N$$

$$Extract\ \mathbf{q}\ the\ k-th\ column\ of\ \mathbf{Q}^T$$

$$Extract\ \mathbf{f}\ the\ k-th\ column\ of\ \mathbf{F} + \mathbf{F}^T - Diag\{\mathbf{F}\}$$

$$Set\ \hat{f}_i = \begin{cases} f_i & i \neq k \\ 2f_k & i = k \end{cases}$$

$$For\ i = 1\ to\ k$$

$$For\ m = i\ to\ N$$

$$S_{mi} \leftarrow S_{mi} - q_i \hat{f}_m$$

$$end\ m$$

$$end\ i$$

$$2.\ \mathbf{S} \leftarrow \mathbf{S} \times \Delta + \sum_{i \ge j} Q_{ij} \frac{\partial^2 f(\mathbf{L})}{\partial \mathbf{L} \partial L_{ij}}$$

$$1.\ Call\ BD(\mathbf{L}, \mathbf{S}, N)$$

$$0.\ \frac{\partial^2 f(L)}{\partial x \partial y} = \sum_{ij} F_{ij} \frac{\partial^2 M_{ij}}{\partial x \partial y} + \sum_{ij} S_{ij} \frac{\partial M_{ij}}{\partial y}$$

The algorithm shown in Display 12 has the desirable property where **Q** can be computed one row or column at a time, with **L** stored in memory. And with **Q** and **F** fully computed, and k-th row-column of each of these can be read into memory one at a time, to compute **S** that is now fully stored in memory.

7. Conclusion

All the algorithms presented in this paper where successfully confirmed by comparing calculated values to known quantities.

The bordering method proved to be a good candidate for the backward differentiation exercises presented in this paper. Not only can the bordering method itself be optimized for efficient applications involving, for example, vector processing, blocking strategies and sparse-matrix manipulation, the key modules needed for the first and second derivatives also lend themselves to optimization. To review, those modules are: GFS or a generalized forward substitution for the **Q** matrix or what is otherwise forward differentiation; GBS or a generalized backward substitution for the **F** matrix or what is otherwise backward differentiation; and Step 3B (of Display 12) for calculating the **S** matrix that is needed for second derivatives. Everything else is a plug-in application of modules, including the appropriate treatment for indefinite matrices (or Δ); i.e., a treatment that fell from being a big challenge and became a minor triviality.

It is helpful to understand the steps involved in factorizing a matrix, or an indefinite matrix, in the large sparse-matrix application. First the matrix Δ is specified, it never changes, and it defines the negative and positive partitions of a matrix **M** that is to be subjected to factorization. The rows and columns must be permuted to permit factorization. This can only be done dynamically using actual floating point numbers and their associated calculations involving the Cholesky decomposition. Unlike the case when **M** is positive definite, a purely symbolic minimum-degree factorization cannot be performed. Factorization with dynamic permutations uses a double linked-list and the outer-product form. Fortunately, this expensive step is done only once. With the permutation order and sparse structure defined, the application turns to repetitive factorization of the matrix **M** evaluated for different parameters, and ths includes repetitive derivative calculations involving functions of **L** (where $\mathbf{L}\,\Delta\,\mathbf{L}^T=\mathbf{M}$). The repetitive steps are meant to utilize optimized software involving the bordering method and its derivatives.

With the permutation order and the sparse structure defined, two integer arrays can be set up that contain this same information. In one array, non-zero elements in the sparse structure are arranged sequentially[9] in column order, before moving to the next row of column identities. In the second array, non-zero elements in the sparse structure are

[9]In terms of the permutation order.

arranged sequentially in row order, before moving to the next column of row identities. The column-arranged array contains the details of column indexes and pointers to the array containing the floating point numbers. The row-arranged array contains the details of row indexes and pointers to the array containing the floating point numbers. It is possible to read through these arrays to perform factorization by the bordering method. Every element encountered in a sequential read corresponds to a non-trivial calculation within the sparse structure. Gone now is any need to search though a linked-list to locate positions in the sparse structure where a calculation applies, as encountered when factorization is by the outer-product form (Ng and Peyton, 1993). The expectation is that these same advantages can be transferred to GFS, GBS and Step 3B[10] of Display 12.

Many statistical applications[11] are found when **M** is positive definite. The factorization by the outer-product form and the calculation of backward derivatives (worked out in 1995) has performed good enough for most cases given today's computer hardware. However, it is always possible to find examples that challenge today's computers, and in which case there is a need to improve the software. An example of a challenging problem is the genetic dominance model (Smith and Mäki-Tanila 1990), and the associated parameter estimation problem. The model is challenging enough to abandon Henderson's (1973) mixed model equations that are popular in animal breeding studies, in preference for a refashioned model that follows Siegel's (1965) equations that have improved sparse-matrix handling properties. For a test example involving egg-laying hens, the parameter estimation problem comes with a very large and sparse **M** matrix of order 330,000; **M** is now indefinite. A permutation order and sparse structure was found using the outer-product form and a double linked-list. The sparse structure ended up containing 24,000,000 non-zero elements. That initial step required 250 minutes of computing time on a desktop computer. Fortunately, with the permutation order and the sparse structure computed, a single application of the bordering algorithm only required 13 minutes of computing time. Work is now continuing to estimate the genetic parameters using the methods described in this paper.

References

De Hoog, R.F., R.S. Anderssen and M.A. Lukas (2011), Differentiation of matrix Functionals using triangular factorization, *Mathematics of Computation,* January, 1-15.

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. (1992), Achieving Logarithmic Growth of Temporal and Spatial Complexity

[10]In particular, Step 3B can mirror the same matrix addressing used for GFS while holding the vector **f** in a hash table, rather than **q** as needed in GFS.

[11]The restricted maximum likelihood (REML) applications, in particular.

in Reverse Automatic Differentiation, *Optimization Methods and Software*, 1, 35-54.

Griewank, A, 2000, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM, Philadelphia, PA

Henderson, C.R., 1973, Sire Evaluation and Genetic Trends, In *Proceeding of the Animal Breeding and Genetics Symposium in Honor of Dr Jay L. Lush*, ASAS and ADSA, Champaign, Illinois, 10-41.

Koerber, P. (2015), Adjoint algorithmic differentiation and the derivatives of the Cholesky decomposition.

Murray, I. (2016), Differentiation of the Cholesky decomposition, arXiv archived.

Ng, E.G. and B.W. Peyton, 1993, Block spare Cholesky algorithms on advanced uniprocessor computers, *SIAM Journal of Scientific Computing*, 14, 1034-1055.

Siegel, I.H., 1965, Deferment of Computation in the Method of Least Squares, *Mathematics of Computation*, 19 (90): 329-331.

Smith, J.R., M. Nikolic and S.P. Smith, 2012, Hunting the Higgs Boson using the Cholesky Decomposition of an Indefinite Matrix, memo, vixRa archived.

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

Smith, S.P., (1995b), The Cholesky decomposition and its derivatives, memo.

Smith, S.P. (2000), A tutorial on simplicity and computational differentiation for statisticians, memo, vixRa archived.

Smith, S.P., 2001a, Factorability of Symmetric Matrices, *Linear Algebra and Its Application*, 335: 63-80.

Smith, S.P., 2001b, Likelihood-Based Analysis of Linear State-Space Models Using the Cholesky Decomposition, *Journal of Computational and Graphical Statistics*, 10 (2): 350-369.

Smith, S.P., 2017, The Bordering Method of the Cholesky Decomposition and Its Backward Differentiation, memo, vixRa archived.

Smith, S.P, and A. Mäki-Tanila, 1990, Genotypic Covariances Matrices and their Inverse for Models Allowing Dominance and Inbreeding, *Genetics Selection Evolution*, 22, 65-91.