The Bordering Method of the Cholesky Decomposition and its Backward Differentiation

Stephen P Smith (hucklebird@aol.com)
February 2017

Abstract

This paper describes the backward differentiation of the Cholesky decomposition by the bordering method. The backward differentiation of the Cholesky decomposition by the inner product form and the outer product form have been described elsewhere. It is found that the resulting algorithm can be adapted to vector processing, as is also true of the algorithms developed from the inner product form and outer product form. The three approaches can also be fashioned to treat sparse matrices, but this is done by enforcing the same sparse structure found for the Cholesky decomposition on a secondary work space.

1. Introduction

There are different ways to order and arrange the calculations of the Cholesky decomposition of an N×N matrix $M=LL^T$. First there are N! ways to permute rows and columns, where some permutations are preferred to maintain sparsity, while others move the largest diagonal element forward to become a pivot and thereby improving numerical stability. Moreover, there are three general ways to perform the Cholesky decomposition (see George and Liu, 1981, Section 2.1.2): the inner product form (or left-looking), the outer product form (or right-looking), and the bordering method. At any of N steps in the algorithm, the calculations can switch over from one of the three general ways to another way (with some transition calculations). A switch over can happen, for example, when a sparse matrix outer product algorithm is used up until a point that fill-in becomes excessive and then a switch over is done into the inner product form that permits parallel vector calculations. Moreover, it is possible to inflate the number of ways to organize the Cholesky decomposition further by adapting software to blocked strategies and partition matrices.

The Cholesky decomposition is not an algorithm that is unambiguously defined without fuller specifications from the choices above. Only with the fuller specifications does it make better sense to strictly speak of the backward differentiation (ref. Griewank, 1989) of the Cholesky decomposition, because different backward differentiation algorithms emerge depending on the selection of computing possibilities. The inner product form, the outer product form, and the bordering method are three methods that come with three different backward differentiations. Partitioning and blocking strategies will also impact on backward differentiation.

Even with the Cholesky decomposition well defined, there is nothing that forces the

forward sweep where L is computed by the selected algorithm, to be followed by the backward sweep of this very same algorithm to calculate the backward derivatives. It is quite possible to use one algorithm to compute L, and to follow this with the backward differentiation of a different algorithm that also does the Cholesky decomposition. For a radical departure, if $M=X^TX$ where X is a rectangular matrix, then L can be computed by the QR algorithm applied to X, and the reverse sweep can be handled by one of the available algorithms for backward differentiation of a more conventional Cholesky decomposition of M. However, even the QR algorithm can be backward differentiated when specified (Walter, Lehmann and Lamour 2011).

Symbolic differentiation can also generate a departure from algorithmic specificity, but this is not necessarily a disadvantage. De Hoog, Anderssen and Lukas (2011), Koerber (2015) and Murray (2016) investigated various symbolic differentiation techniques that utilize powerful notations to differentiate the Cholesky decomposition, or the LU factorization, all leading to algorithms that resemble forward and reverse mode differentiation of the Cholesky decomposition. A closer study of symbolic differentiation is presented in the Appendix. Giles (2008) provides a very useful collection of forward and backward derivatives for common matrix operations and are ready-made for symbolic differentiation, but when it came to the Cholesky decomposition Giles performed straight backward differentiation on the outer product form. These are variants that lead to hand-coding, where symbolism is turned into a computer program by a programer, thereby avoiding automatic differentiation by software. Hand-coding is symbolic differentiation applied on a fine scale, including backward differentiation or one of the other variants that come from powerful notation. It is only symbolic differentiation of multi-variate expressions that leads to lost algorithmic specificity.

Smith (1995a) performed the backward differentiation of the outer product form of the Cholesky decomposition before Giles, and fashioned that algorithm for sparse matrices. Murray (2016) performed backward differentiation on the inner product form. Murray was more interested in dense matrices, and so Murray fashioned the approach to vector processing, including the blocked Cholesky algorithm.

The backward differentiation of the bordering method has been unexplored or not publicized. It is the purpose of this note to provide that differentiation so that all three versions are available.

2. Bordering Method for the Cholesky Decomposition

To introduce the bordering method the following definitions are required.

$$A_{k+1} = \begin{bmatrix} A_k & a_{k+1} \\ a_{k+1}^T & \alpha_{k+1} \end{bmatrix} = \begin{bmatrix} L_k & \\ u_{k+1}^T & d_{k+1} \end{bmatrix} \begin{bmatrix} L_k^T & u_{k+1} \\ & d_{k+1} \end{bmatrix} = L_{k+1}^T L_{k+1}$$

where:

$$A_k = L_k L_k^T$$
$$u_{k+1} = L_k^{-1} a_{k+1}$$
$$d_{k+1} = \sqrt{\alpha_{k+1} - u_{k+1}^T u_{k+1}}$$

It is noted that the lower triangular matrix L_k is the Cholesky decomposition of A_k, recursively for k=1, 2,... N, where A_1 and L_1 are the scalars α_1 and $d_1=\alpha_1^{½}$, respectively. It is noted that the lower triangular matrix L_{k+1} is cobbled together from L_k, u_{k+1} and d_{k+1}. The series of matrices A_1, A_2, ... A_N, are the leading sub-matrices of $A_N=M$, an N x N symmetric and positive definite matrix with the Cholesky decomposition $L_N^T L_N=M$.

The bordering method to compute the Cholesky decomposition of the matrix M is given below.

1. Set $A_N=M$, thereby defining all the arrays, A_k, a_k, α_k, $N \geq k>1$, and $A_1=\alpha_1$, implicitly as data entries.

2. Evaluate $d_1=\alpha_1^{½}$.

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

    3a. Solve u_{k+1} in the lower triangular system, $L_k u_{k+1}=a_{k+1}$, by forward substitution.

    3b. Evaluate the vector product, $\xi=u_{k+1}^T u_{k+1}$, and then evaluate $d_{k+1}=(\alpha_{k+1} - \xi)^{½}$.

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 L_N while following the bordering method.

3. Backward Differentiation of the Bordering Method

Because the bordering method is highly granular, where all intermediate calculations are saved, it need not involve overwriting. Except for the possibility of overwriting various parts of the initial data or the matrix M with various parts of L_N as they are

computed, the application of the *Rules for Backward Differentiation*[1] are most transparent. Even when matrix M is overwritten, enough information is saved in 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 $F_{L(N)}$ correspond to all the non-data intermediates that are all neatly collected in L_N and let 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 $F_{L(N)} \leftarrow F_M$ in the backward sweep where F_M signifies a half-stored matrix. The symbolic representations preserve the distinction between $F_{L(N)}$ and F_M , even though $F_{L(N)}$ is lost by overwriting, and that is enough for our purpose. Likewise, let $F_{L(k)}$, $F_{u(k)}$, and $F_{d(k)}$ correspond to the intermediates of L_k, u_k and d_k, all belonging to the larger array $F_{L(N)}$. Let $F_{A(k)}$, $F_{a(k)}$ and $F_{\alpha(k)}$ correspond to A_k, a_k, α_k, all belonging to F_M.

The only other adjustment to the Rules for Backward Differentiation that is required has to do with the $h_k=f_k(h_i)$ function (see footnote 1) that is meant to represent step 3a (above) that is found to be a non-scalar. Therefore, the needed update involves vector multiplication of the row vector $F_{h(k)}^T$ and column vector $\partial h_k/\partial h_i$ for $h_i \in \Omega_k$.

The Rules for Backward Differentiation when applied to the bordering method above gives the following algorithm.

A. Set F_M = null, and $F_{L(N)} = \partial h(L_N)/\partial L_N$, where $h() = h_r()$ is the scalar function at the end of the recursion list of length r (see footnote 1).

For k=N-1 to 1, perform steps B, C, D, and F below.

For step 3b calculations:

B. $F_{\alpha(k+1)} = F_{d(k+1)}/d_{k+1}$

C. $F_{u(k+1)}=F_{u(k+1)} - u_k F_{\alpha(k+1)}$

D. $F_{\alpha(k+1)} = \frac{1}{2} F_{\alpha(k+1)}$

---

[1]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. The abbreviated version follows: 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.

For step 3a calculations:

E. $F_{a(k+1)}{}^T = F_{u(k+1)}{}^T L_k^{-1}$, and done more frugally by solving $F_{a(k)}$ in the upper triangular system, $L_k{}^T F_{a(k+1)} = F_{u(k+1)}$, by backward substitution.

F. Make the following adjustments on $F_{L(k)}$ where $F_{L(k)-ij}$ is the ij-th element of $F_{L(k)}$, and $j \le i$. Set $F_{L(k)-ij} = F_{L(k)-ij} - F_{u(k+1)}{}^T L_k^{-1} [e_i \times e_j{}^T] L_k^{-1} a_{k+1}$ for all $j \le i \le k-1$, where e_i is a null column vector except for the i-th position that is set to 1. This set of adjustments can be done more frugally by setting $F_{L(k)} = F_{L(k)} - \text{Lower}(F_{a(k+1)} \times u_{k+1}{}^T)$, where Lower() is a matrix function that returns a lower triangular matrix that is extracted from $F_{a(k+1)} \times u_{k+1}{}^T$ while intervening on the vector product to avoid computing elements above the diagonal.

G. Apply the above for k>0 and when done perform the last calculation that corresponds to k=0, $F_{\alpha(1)} = \frac{1}{2} F_{d(1)}/d_1$.

The more frugal overwriting is easily recognize where $F_{L(N)} \leftarrow F_M$, where all the steps can rewritten to reflect overwriting. However, it is better to first recognize that steps B, C and G can be made part of the backward substitution conducted during step E but as part of a larger system, with step D postponed until after step F, while changing the summation to range over k from N to 1. With these changes the algorithm is more neatly expressed below.

A★. Set $F_M = \partial h(L_N)/\partial L_N$, where $h() = h_r()$ is the scalar function at the end of the recursion list of length r.

For k=N to 1, perform steps below.

B★. For k>1, apply $\{F_{a(k)}{}^T, F_{\alpha(k)}\} \leftarrow \{F_{a(k)}{}^T, F_{\alpha(k)}\} L_k^{-1}$, and done more frugally by using backward substitution to solve the k×1 column vector v_k in upper triangular system, $L_k{}^T v_k = \{F_{a(k)}{}^T, F_{\alpha(k)}\}^T$, but overwriting $F_{a(k)}$ and $F_{\alpha(k)}$ with v_k in place. When k=1, this becomes a trivial adjustment: $F_{\alpha(1)} \leftarrow F_{\alpha(1)}/d_1$.

C★. Make the following adjustments to the half-stored matrix F_A, $F_{A(k-1)} \leftarrow F_{A(k-1)} - \text{Lower}(F_{a(k)} \times u_k{}^T)$, if k>1.

D★. $F_{\alpha(k)} \leftarrow \frac{1}{2} F_{\alpha(k)}$

It is possible to postpone all the step C★ adjustments, until when they are needed. The next iteration requires that some of these adjustment must be applied before $F_{a(k-1)}$ and $F_{\alpha(k-1)}$ can be updated, if k>1. These particular adjustments are of the form, $\{F_{a(k-1)}{}^T, F_{\alpha(k-1)}\}^T \leftarrow \{F_{a(k-1)}{}^T, F_{\alpha(k-1)}\}^T - B_{k-1}{}^T w_{k-1}$ where B_k is a rectangular sub-matrix of L that is located in the lower left corner where the first k rows and the last N-k columns are stricken, and w_k is a column vector out of F_A that is strictly below the k-th diagonal and has already been computed. These operations can be neatly appended to step B★, so that they are first up during a generalized backward substitution, noting that $\{L_k{}^T, B_k{}^T\}^T$ is

a proper sub-matrix of L, and $\{v_k^T, w_k^T\}^T$ becomes the k-th column of the full-stored version of F_A once v_k is computed.[2]

A numerical test found that the above algorithm (involving A⋆, B⋆, C⋆ and D⋆) gave the correct calculation when compared to results found symbolically. When step C⋆ was postponed as described above, the correct calculation was also retrieved.

4. Discussion

The backward differentiation of the bordering method (Section 3) as structured can benefit from vector processing, with all of the $o(N^3)$ operations coming from steps B⋆ and C⋆:

B⋆. Use backward substitution to solve the k×1 column v_k in upper triangular system, $L_k^T v_k = \{F_{a(k)}^T, F_{\alpha(k)}\}^T$, but overwriting $F_{a(k)}$ and $F_{\alpha(k)}$ with v_k in place.

C⋆. $F_{A(k-1)} \leftarrow F_{A(k-1)} - \text{Lower}(F_{a(k)} \times u_k^T)$.

Vector processing is feasible because each of the step C⋆ adjustments of $F_{A(k-1)}$ can be done independently, and because most of the operations in step B⋆, those involving subtraction, can be done independently in k groups. There still might be a burden that creates a relative inefficiency due to accessing quantities stored in memory (cashe), and therefore bench-mark testing is recommended when comparing different algorithms. When making comparisons, it makes no difference which algorithm is used during the forward sweep, as explained in Section 1.

Murray (2016) fashioned the backward differentiation on the inner product form to vector processing, including a blocked Cholesky algorithm. While Smith (1995) did not consider the feasibility of blocking or vector processing to perform the backward differentiation of the outer product form, it is clear that all of the $o(N^3)$ operations can also be put into groups that are done independently, because they involve matrix and vector multiplications representing the reverse differentiation of a rank-1 update:

1. Rank-1 update during forward sweep of a symmetric matrix H, $H \leftarrow H - ww^T$.

2. Hence, the reverse sweep is of the form, $F_w \leftarrow F_w - [\text{Lower}(F_H) + \text{Lower}(F_H)^T]\, w$, where F_H is also a symmetric matrix.

Therefore, the backward differentiation the Cholesky decomposition can benefit from

[2]The full stored version of F_A is $F_A + F_A^T - \text{Diag}(F_A)$, where the matrix function Diag() is defined in the Appendix.

vector processing (if not a fully blocked algorithm) for the bordering method, the inner product form, and the outer product form.

The forward sweeps of the bordering method, the inner product form, and the outer product form can all be fashioned to exploit sparse matrix structure (George and Liu, 1981). Smith (1995) found that the array F_L has the same sparse structure as L, a result that was transparent from the differentiation of the outer product form by the reverse sweep. Therefore, it is feasible in principle to also fashion the backward derivatives of the bordering method and the inner product form to sparse matrix manipulation, by enforcing that F_L and L share the same sparse structure. However, whenever symbolic differentiation is used to guide hand programming on multi-variate functions, it is possible to loose sight of the sparse structure of F_L. Symbolic differentiation does enter into the backward differentiation of the bordering method and to a lesser extent the inner product form. In the case of the bordering method of Section 2, step 3a is a multi-variate function and it results in the steps B$\star$ and C$\star$ (Section 3). The backward differentiation of forward substitution (step 3a, Section 2) does product the backward substitution found in step B$\star$ exactly[3], and step C$\star$ is given symbolically as,

$$F_{A(k-1)} \leftarrow F_{A(k-1)} - \text{Lower}(F_{a(k)} \times u_k^T)$$

but this will destroy sparsity if applied blindly. In fact, the introduction of the matrix function Lower() was done to enforce the lower triangular structure of L and F, but it is not part of the symbolic derivatives of $u_k=L_k^{-1}a_k$ given by Giles (2008), if we lose sight of the fact that L_k is lower triangular. Ironically, the update is valid even if the function Lower() is removed, but those extra derivatives are unwanted. It's the same way with the general sparse structure. When faithfully going through the backward differentiation (of forward substitution) that produces the backward substitution that is required for step B$\star$, it is also found that the step C$\star$ update involving $F_{A(k-1)}$ only occurs if the corresponding element of L is occupied and not set to zero within the sparse structure.

Even though the inner product form, the outer product form and the bordering method are only different because of the different ordering of calculations for the Cholesky decomposition, these relate differentially to computational strategies for treating sparsity, blocking and the utilization of vector processing where some approaches are found efficient and others not so much (Ng and Peyton 1993). Moreover, when backward differentiation is applied to the three ways to calculate the Cholesky decomposition, what is found is not a trivial reordering of calculations, what comes out is a notable complexification[4]. The three different algorithms, or organizations, should

[3]Apart from an inconsequential reordering of calculations.

[4]If backward substitution involving a triangular linear system is vaguely defined to be a "flipped" forward substitution, then by analogy the backward differentiation of the outer product

be tested anew for treating sparsity, blocking and the utilization of vector processing, and done quite independently to what works well for the forward sweep. However, the noted complexification may inform our expectations of test results.

References

Brewer, J.W. (1977), The gradient with respect to a symmetric matrix, IEEE Transactions on Automatic Control, April, 265-267.

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

George, A. and J.W. Liu (1981), Computer Solutions of Large Sparse Positive Definite Systems, Pretice-Hall, Inc, Englewood Cliffs, New Jersey.

Giles, M. (2008), An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation, Oxford University Computing Laboratory, Report no. 08/01.

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.

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.

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.

---

form becomes a flipped bordering method, the backward differentiation of the boarding method becomes a flipped outer product form, and the backward differentiation of the inner product form becomes a flipped inner product form.

Walter, S.F, L. Lehmann and R. Lamour (2011), On Evaluating higher-order derivatives of the QR decomposition of tall matrices with full column rank in forward and reverse mode algorithmic differentiation, Optimization Methods & Software, 1-13.

Appendix: Symbolic Differentiation of the Cholesky Decomposition

While powerful notation is available to symbolically differentiate the Cholesky decomposition, and to derive the backward derivative update, the published derivations are complex and hard to follow once they turn to the reverse-mode derivative update. Therefore, a detailed derivation of these results is helpful with the expressed purpose of avoiding further layers of complexity and to simplify the derivations as much as possible. Attention is restricted to the Cholesky decomposition, and not the LU factorization. Originality for the following is not claimed.

M is an N×N symmetric matrix, where $M=LL^T$ and L is a lower triangular.

Definitions that describe differentiation with respect to a scalar x:

$$\frac{\partial M}{\partial x} = M_x$$
$$\frac{\partial L}{\partial x} = L_x$$

where M_x is a symmetric matrix representing the derivatives of M in place, likewise L_x is a lower triangular matrix representing the derivatives of L in place.

Therefore, differentiating M by the chain rule reveals the following.

$M_x=L_xL^T + LL_x^T$

Pre-multiplying this equation by L^{-1}, and post-multiplying by L^{-1T}, results in the following.

(1) $\quad L^{-1} M_x L^{-1T}= L^{-1} L_x + L_x^T L^{-1T}$

The right-hand side is revealed to be the sum of a lower triangular matrix, $L^{-1} L_x$ and an upper triangular matrix, $L_x^T L^{-1T}$.

Define the matrix function Φ(A) that extracts a lower triangular matrix from the N×N matrix A:

$$\Phi(A) = \begin{cases} \frac{1}{2}A_{ij}, if\ j=i \\ A_{ij},\ if\ j<i \\ 0,\ if\ j>i \end{cases}$$

Define the following matrix functions:

$$Diag(A)=\begin{cases}A_{ij}, if\ j=i\\ 0, \ \ if\ j<i\\ 0, \ \ if\ j>i\end{cases}\ Lower(A)=\begin{cases}A_{ij}, \ \ if\ j=i\\ A_{ij}, if\ j<i\\ 0, \ \ if\ j>i\end{cases}\ Upper(A)=\begin{cases}0, \ \ if\ j=i\\ 0, \ \ if\ j<i\\ A_{ij}, if\ j>i\end{cases}$$

Then, A= Lower(A)+Upper(A), and $\Phi(A)$=A-½Diag(A)-Upper(A)=Lower(A)-½Diag(A) .

With this notation, (1) implies $\Phi(L^{-1} M_x L^{-1T}) = L^{-1} L_x$. Solving for L_x reveals the symbolic derivatives of L with respect to x:

(2) $\quad L_x = L\ \Phi(L^{-1} M_x L^{-1T})$

Murray's (2016) derivation of (2) is the same as the above. De Hoog, Anderssen and Lukas's (2011) gave a very similar derivation so far, but for the more general case involving the LU factorization.

If f(M,L) is a scalar function of M and L, then to accumulate the backward derivatives, or sensitivities, following the Rules for Backward Differentiation (see footnote 1), what is needed is an array F representing the elements of M and L. Let F_M represent the elements corresponding to M, F_L the elements corresponding to L, $F_{M(i,j)}$ the element corresponding the ij-the element of M, and $F_{L(i,j)}$ the element corresponding to the ij-th element of L ($j \le i$).

To begin backward differentiation, make the following initializations.

$F_M = \partial f(M,L)/\partial M$

$F_L = \partial f(M,L)/\partial L$

Then the following updates are applied for all $j \le i$.

$F_{M(i,j)} = F_{M(i,j)} + tr[F_L^T\ \partial L/\partial M_{ij}]$

The matrix of partial derivatives of L with respect to M_{ij}, i.e., $\partial L/\partial M_{ij}$, is provided by equation (2) with $x=M_{ij}$ and $M_x= M_{M(i,i)} = e_i e_j^T+e_j e_i^T$ (if $i \ne j$) or $e_i e_j^T$ (if i=j) following Brewer (1977), where e_i is a null column vector except of an entry one at the i-th position. If the initialization for $F_{M(i,j)}$ was set to null, because f() was not selected to be a direct function of M as is the usual case, the aforementioned substitutions produce the update equations below.

(3) $\quad F_{M(i,j)} = tr[F_L^T\ L\ \Phi(L^{-1} M_{M(i,i)} L^{-1T})\ ]$, for all $j \le i$.

To evaluate (3), consider the simpler trace calculation given by (4).

(4) $G_{ij} = tr[F_L^T \, L \, \Phi(L^{-1} \, e_i \, e_j^T \, L^{-1T}) \,]$

If the right-hand side of (4) can be expressed as the ij-th element of a matrix G, then the update equation (3) becomes (5).

(5) $F_M = Lower(G + G^T) - Diag(G)$

To evaluate (4), define $\omega_i = L^{-1} e_i$, as the i-th column of L^{-1}. Generalize the function Diag() to also operate on vectors, such that Diag(ω) is a diagonal matrix with i-th diagonal being the i-th element of ω. Therefore, Diag(ω) **1** = ω, where **1** is understood to be a column vector of only the number one for its entries. The function $\Phi(\omega_i \, \omega_j^T)$ contained in (4) is rewritten as,

$$\begin{aligned}\Phi(\omega_i \, \omega_j^T) = \Phi(Diag(\omega_i) \, \mathbf{1} \times \mathbf{1}^T \, Diag(\omega_j) \,) &= Diag(\omega_i) \times \Phi(\mathbf{1} \times \mathbf{1}^T) \times Diag(\omega_j) \\ &= Diag(\omega_i) \times \Phi(J) \times Diag(\omega_j) \\ &= Diag(\omega_i) \times K \times Diag(\omega_j)\end{aligned}$$

where J is a matrix of the number one for its entries, as is evident from $J=\mathbf{1} \times \mathbf{1}^T$, and K is the lower triangular matrix defined by $K=\Phi(J)$. Note that diagonal matrices can be factored out of the matrix function, Φ(), on both sides.

Equation (4) beomes

(6) $G_{ij} = tr[\, F_L^T \times L \times Diag(\omega_i) \times K \times Diag(\omega_j) \,] = tr[\, Diag(\omega_j) \times F_L^T \times L \times Diag(\omega_i) \times K \,]$

Defining $B = Diag(\omega_j) \times F_L^T \times L \times Diag(\omega_i)$, note that the right-hand side of (6) is of the form tr[B K]. It is apparent that $tr[B \, K] = \mathbf{1}^T (B^T \bullet K) \mathbf{1}$, where $B^T \bullet K$ is the direct product of B^T on K (or the entry by entry multiplication). However, note that $B^T \bullet K = \Phi(B^T)$, therefore (6) becomes the following.

$$\begin{aligned}(7) \quad G_{ij} = \mathbf{1}^T \Phi(B^T) \, \mathbf{1} = \mathbf{1}^T \Phi(Diag(\omega_i) \, L^T \, F_L \, Diag(\omega_j)) \, \mathbf{1} &= \mathbf{1}^T \, Diag(\omega_i) \, \Phi(L^T \, F_L) \, Diag(\omega_j) \, \mathbf{1} \\ &= \omega_i^T \, \Phi(L^T \, F_L) \, \omega_j \\ &= e_i^T \, L^{-1 \, T} \, \Phi(L^T \, F_L) \, L^{-1} \, e_j\end{aligned}$$

Now the ij-th element of G can be read directly from (7), where it is apparent that $G = L^{-1T} \, \Phi(L^T \, F_L) \, L^{-1}$, and update equation (5) can now be expressed in the handy form.

(8) $F_M = Lower(L^{-1T} \{\Phi(L^T \, F_L) + \Phi(L^T \, F_L)^T\} \, L^{-1}) - Diag(L^{-1T} \, \Phi(L^T \, F_L) \, L^{-1})$

Murray derived a reverse derivative update that agrees exactly with (8). A vague resemblance was found to (8) following Koerber's (2015) very different derivation, with exact agreement found following from the matrix relation $\Phi(A^T)^T = ½Diag(A) + Upper(A)$ for any matrix A. While De Hoog, Anderssen and Lukas's (2011) treatment was for the

more general LU factorization, exact agreement is again found with (8) by substituting the upper triangular matrix U in their equation (3.13) with $U=L^T$. Hand programming from these various symbolically derived results can precede and might even lead to different programs, in addition to what is expected from the lack of algorithmic specificity.