Report no. 08/01

An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation

Mike Giles
Oxford University Computing Laboratory, Parks Road, Oxford, U.K.

This paper collects together a number of matrix derivative results which are very useful in forward and reverse mode algorithmic differentiation (AD). It highlights in particular the remarkable contribution of a 1948 paper by Dwyer and Macphail which derives the linear and adjoint sensitivities of a matrix product, inverse and determinant, and a number of related results motivated by applications in multivariate analysis in statistics.

This is an extended version of a paper which will appear in the proceedings of AD2008, the 5th International Conference on Automatic Differentiation.

Key words and phrases: algorithmic differentiation, linear sensitivity analysis, numerical linear algebra

Oxford University Computing Laboratory
Numerical Analysis Group
Wolfson Building
Parks Road
Oxford, England OX1 3QD

January, 2008

1 Introduction

As the title suggests, there are very few, if any, new results in this paper. Instead, it is a collection of results on derivatives of matrix functions, expressed in a form suitable for both forward and reverse mode algorithmic differentiation [7] of basic operations in numerical linear algebra. All results are derived from first principles, and it is hoped this will be a useful reference for the AD community.

The first section in the paper covers the sensitivity analysis for matrix product, inverse and determinant, and other associated results. Remarkably, most of these results were first derived, although presented in a slightly different form, in a 1948 paper by Dwyer and Macphail [4]. Comments in a paper by Dwyer in 1967 [3] suggest that the "Dwyer/Macphail calculus" was not widely used in the intervening period, but thereafter it has been used extensively within statistics, appearing in a number of books [10, 13, 14, 16] from the 1970's onwards. For a more extensive bibliography, see the notes at the end of section 1.1 in [11]. The section concludes with a discussion of Maximum Likelihood Estimation which was one of the motivating applications for Dwyer's work, and comments on how the form of the results in Dwyer and Macphail's paper relates to the AD notation used in this paper.

The subsequent sections concern the sensitivity of eigenvalues and eigenvectors, singular values and singular vectors, Cholesky factorisation, and associated results for matrix norms. The main linear sensitivity results are well established [10, 17]. Some of the reverse mode adjoint sensitivities may be novel, but they follow very directly from the forward mode linear sensitivities. The paper concludes with a validation of the mathematical results using a MATLAB code which is given in the appendix.

2 Matrix product, inverse and determinant

2.1 Preliminaries

We consider a computation which begins with a single scalar input variable S_I and eventually, through a sequence of calculations, computes a single scalar output S_O. Using standard AD terminology, if A is a matrix which is an intermediate variable within the computation, then $\dot{A}$ denotes the derivative of A with respect to S_I, while $\overline{A}$ (which has the same dimensions as A, as does $\dot{A}$) denotes the derivative of S_O with respect to each of the elements of A.

Forward mode AD starts at the beginning and differentiates each step of the computation. Given an intermediate step of the form

$$C = f(A, B)$$

then differential calculus expresses infinitesimal perturbations to this as

$$\mathrm{d}C = \frac{\partial f}{\partial A}\,\mathrm{d}A + \frac{\partial f}{\partial B}\,\mathrm{d}B. \tag{2.1}$$

Taking the infinitesimal perturbations to be due to a perturbation in the input variable S_I gives

$$\dot{C} = \frac{\partial f}{\partial A}\,\dot{A} + \frac{\partial f}{\partial B}\,\dot{B}.$$

This defines the process of forward mode AD, in which each computational step is differentiated to determine the sensitivity of the output to changes in S_I.

Reverse mode AD computes sensitivities by starting at the end and working backwards. By definition,

$$\mathrm{d}S_O = \sum_{i,j} \overline{C}_{i,j}\,\mathrm{d}C_{i,j} = \mathrm{Tr}(\,\overline{C}^T\mathrm{d}C),$$

where $\mathrm{Tr}(A)$ is the trace operator which sums the diagonal elements of a square matrix. Inserting (2.1) gives

$$\mathrm{d}S_O = \mathrm{Tr}\left(\overline{C}^T\frac{\partial f}{\partial A}\,\mathrm{d}A\right) + \mathrm{Tr}\left(\overline{C}^T\frac{\partial f}{\partial B}\,\mathrm{d}B\right).$$

Assuming A and B are not used in other intermediate computations, this gives

$$\overline{A} = \left(\frac{\partial f}{\partial A}\right)^T\overline{C}, \qquad \overline{B} = \left(\frac{\partial f}{\partial B}\right)^T\overline{C}.$$

This defines the process of reverse mode AD, working backwards through the sequence of computational steps originally used to compute S_O from S_I. The key therefore is the identity

$$\mathrm{Tr}(\,\overline{C}^T\mathrm{d}C) = \mathrm{Tr}(\,\overline{A}^T\mathrm{d}A) + \mathrm{Tr}(\,\overline{B}^T\mathrm{d}B). \tag{2.2}$$

To express things in this desired form, the following identities will be useful:

$$\begin{aligned}
\mathrm{Tr}(A^T) &= \mathrm{Tr}(A),\\
\mathrm{Tr}(A{+}B) &= \mathrm{Tr}(A) + \mathrm{Tr}(B),\\
\mathrm{Tr}(A\,B) &= \mathrm{Tr}(B\,A).
\end{aligned}$$

In considering different operations $f(A, B)$, in each case we first determine the differential identity (2.1) which immediately gives the forward mode sensitivity, and then manipulate it into the adjoint form (2.2) to obtain the reverse mode sensitivities. This is precisely the approach used by Minka [12] (based on Magnus and Neudecker [10]) even though his results are not expressed in AD notation, and the reverse mode sensitivities appear to be an end in themselves, rather than a building block within an algorithmic differentiation of a much larger algorithm.

2.2 Elementary results

2.2.1 Addition

If $C = A + B$ then obviously

$$\mathrm{d}C = \mathrm{d}A + \mathrm{d}B$$

and hence in forward mode

$$\dot{C} = \dot{A} + \dot{B}.$$

Also,

$$\mathrm{Tr}(\overline{C}^T \mathrm{d}C) = \mathrm{Tr}(\overline{C}^T \mathrm{d}A) + \mathrm{Tr}(\overline{C}^T \mathrm{d}B)$$

and therefore in reverse mode

$$\overline{A} = \overline{C}, \qquad \overline{B} = \overline{C}.$$

2.2.2 Multiplication

If $C = A\,B$ then

$$\mathrm{d}C = \mathrm{d}A\ B + A\ \mathrm{d}B$$

and hence in forward mode

$$\dot{C} = \dot{A}\,B + A\,\dot{B}.$$

Also,

$$\mathrm{Tr}(\,\overline{C}^T \mathrm{d}C) \ = \ \mathrm{Tr}(\,\overline{C}^T \mathrm{d}A\,B) + \mathrm{Tr}(\,\overline{C}^T A\,\mathrm{d}B) \ = \ \mathrm{Tr}(B\,\overline{C}^T \mathrm{d}A) + \mathrm{Tr}(\,\overline{C}^T A\,\mathrm{d}B),$$

and therefore in reverse mode

$$\overline{A} = \overline{C}\,B^T, \qquad \overline{B} = A^T\,\overline{C}.$$

2.2.3 Inverse

If $C = A^{-1}$ then

$$C\ A = I \quad \Longrightarrow \quad \mathrm{d}C\,A + C\,\mathrm{d}A = 0 \quad \Longrightarrow \quad \mathrm{d}C = -\,C\ \mathrm{d}A\ C.$$

Hence in forward mode we have

$$\dot{C} = -\,C\,\dot{A}\,C.$$

Also,

$$\mathrm{Tr}(\,\overline{C}^T \mathrm{d}C) \ = \ \mathrm{Tr}(-\overline{C}^T A^{-1} \mathrm{d}A\,A^{-1}) \ = \ \mathrm{Tr}(-A^{-1}\overline{C}^T A^{-1} \mathrm{d}A)$$

and so in reverse mode

$$\overline{A} = -A^{-T}\,\overline{C}\,A^{-T} = -C^T\,\overline{C}\,C^T.$$

2.2.4 Determinant

If we define $\widetilde{A}$ to be the matrix of co-factors of A, then

$$\det A = \sum_j A_{i,j}\widetilde{A}_{i,j}, \qquad A^{-1} = (\det A)^{-1}\widetilde{A}^T.$$

for any fixed choice of i. If $C = \det A$, it follows that

$$\frac{\partial C}{\partial A_{i,j}} = \widetilde{A}_{i,j} \quad \Longrightarrow \quad \mathrm{d}C \;=\; \sum_{i,j} \widetilde{A}_{i,j}\,\mathrm{d}A_{i,j} \;=\; C\ \mathrm{Tr}(A^{-1}\mathrm{d}A).$$

Hence, in forward mode we have

$$\dot{C} = C\ \mathrm{Tr}(A^{-1}\dot{A}),$$

while in reverse mode C and $\overline{C}$ are both scalars and so we have

$$\overline{C}\ \mathrm{d}C \;=\; \mathrm{Tr}(\,\overline{C}\,C\,A^{-1}\mathrm{d}A)$$

and therefore

$$\overline{A} = \overline{C}\,C\,A^{-T}.$$

Note: in a paper in 1994 [9], Kubota states that the result for the determinant is well known, and explains how reverse mode differentiation can therefore be used to compute the matrix inverse.

2.3 Additional results

Other results can be obtained from combinations of the elementary results.

2.3.1 Matrix inverse product

If $C = A^{-1}B$ then

$$\mathrm{d}C = \mathrm{d}A^{-1}\,B + A^{-1}\,\mathrm{d}B = -A^{-1}\mathrm{d}A\,A^{-1}B + A^{-1}\,\mathrm{d}B = A^{-1}(\mathrm{d}B - \mathrm{d}A\,C),$$

and hence

$$\dot{C} = A^{-1}(\dot{B} - \dot{A}\,C),$$

and

$$\begin{aligned}
\mathrm{Tr}(\,\overline{C}^T\mathrm{d}C) \;&=\; \mathrm{Tr}(\,\overline{C}^T A^{-1}\mathrm{d}B) - \mathrm{Tr}(\,\overline{C}^T A^{-1}\mathrm{d}A\,C) \\
&=\; \mathrm{Tr}(\,\overline{C}^T A^{-1}\mathrm{d}B) - \mathrm{Tr}(C\,\overline{C}^T A^{-1}\mathrm{d}A) \\
\Longrightarrow \qquad \overline{B} = A^{-T}\,\overline{C}, &\qquad \overline{A} = -A^{-T}\,\overline{C}\,C^T = -\overline{B}\,C^T.
\end{aligned}$$

2.3.2 First quadratic form

If $C = B^T A\, B$, then

$$\mathrm{d}C = \mathrm{d}B^T A\, B + B^T \mathrm{d}A\, B + B^T A\, \mathrm{d}B.$$

and hence

$$\dot{C} = \dot{B}^T A\, B + B^T \dot{A}\, B + B^T A\, \dot{B},$$

and

$$\begin{aligned} \mathrm{Tr}(\,\overline{C}^T \mathrm{d}C) &= \mathrm{Tr}(\,\overline{C}^T \mathrm{d}B^T A\, B) + \mathrm{Tr}(\,\overline{C}^T B^T \mathrm{d}A\, B) + \mathrm{Tr}(\,\overline{C}^T B^T A\, \mathrm{d}B) \\ &= \mathrm{Tr}(\,\overline{C} B^T A^T\, \mathrm{d}B) + \mathrm{Tr}(B\, \overline{C}^T B^T \mathrm{d}A) + \mathrm{Tr}(\,\overline{C}^T B^T A\, \mathrm{d}B) \\ \Longrightarrow \qquad & \overline{A} = B\, \overline{C}\, B^T, \qquad \overline{B} = A\, B\, \overline{C}^T + A^T\, B\, \overline{C}. \end{aligned}$$

2.3.3 Second quadratic form

If $C = B^T A^{-1} B$, then similarly one gets

$$\dot{C} = \dot{B}^T A^{-1} B - B^T A^{-1} \dot{A}\, A^{-1} B + B^T A^{-1} \dot{B},$$

and

$$\overline{A} = -A^{-T} B\, \overline{C}\, B^T A^{-T}, \qquad \overline{B} = A^{-1} B\, \overline{C}^T + A^{-T} B\, \overline{C}.$$

2.3.4 Matrix polynomial

Suppose $C = p(A)$, where A is a square matrix and $p(A)$ is the polynomial

$$p(A) = \sum_{n=0}^{N} a_n A^n.$$

Pseudo-code for the evaluation of C is as follows:

$C := a_N I$
`for` n `from` $N-1$ `to` 0
    $C := A\, C + a_n I$
`end`

where I is the identity matrix with the same dimensions as A.

Using standard forward mode AD with the matrix product results gives the corresponding pseudo-code to compute $\dot{C}$:

$\dot{C} := 0$
$C := a_N I$
`for` n `from` $N-1$ `to` 0
    $\dot{C} := \dot{A}\,C + A\,\dot{C}$
    $C := A\,C + a_n I$
`end`

Similarly, the reverse mode pseudo-code to compute $\overline{A}$ is:

$C_N := a_N I$
`for` n `from` $N-1$ `to` 0
    $C_n := A\,C_{n+1} + a_n I$
`end`
$\overline{A} := 0$
`for` n `from` 0 `to` $N-1$
    $\overline{A} := \overline{A} + \overline{C}\,C_{n+1}^T$
    $\overline{C} := A^T\,\overline{C}$
`end`

Note the need in the above code to store the different intermediate values of C in the forward pass so that they can be used in the reverse pass. This storage requirement is standard in reverse mode computations [7].

2.3.5 Matrix exponential

In MATLAB, the matrix exponential

$$\exp(A) \equiv \sum_{n=0}^{\infty} \frac{1}{n!} A^n,$$

is approximated through a scaling and squaring method as

$$\exp(A) \approx \Big(p_1(A)^{-1} p_2(A)\Big)^m,$$

where m is a power of 2, and p_1 and p_2 are polynomials such that $p_2(x)/p_1(x)$ is a Padé approximation to $\exp(x/m)$ [8]. The forward and reverse mode sensitivities of this approximation can be obtained by combining the earlier results for the matrix inverse product and polynomial.

2.4 MLE and the Dwyer/Macphail paper

A d-dimensional multivariate Normal distribution with mean vector μ and covariance matrix Σ has the joint probability density function

$$p(x) = \frac{1}{\sqrt{\det \Sigma}\ (2\pi)^{d/2}}\ \exp\Big(-\tfrac{1}{2}(x-\mu)^T\,\Sigma^{-1}(x-\mu)\Big)\,.$$

Given a set of N data points x_n, their joint probability density function is

$$P = \prod_{n=1}^{N} p(x_n).$$

Maximum Likelihood Estimation infers the values of μ and Σ from the data by choosing the values which maximise P. Since

$$\log P = \sum_{n=1}^{N} \Big\{ -\tfrac{1}{2}\log(\det\Sigma) - \tfrac{1}{2}d\,\log(2\pi) - \tfrac{1}{2}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)\Big\}\,,$$

the derivatives with respect to μ and Σ are

$$\frac{\partial \log P}{\partial \mu} = -\sum_{n=1}^{N} \Sigma^{-1}(x_n-\mu),$$

and

$$\frac{\partial \log P}{\partial \Sigma} = -\tfrac{1}{2}\sum_{n=1}^{N} \Big\{\Sigma^{-1} - \Sigma^{-1}(x_n-\mu)\,(x_n-\mu)^T\Sigma^{-1}\Big\}\,.$$

Equating these to zero gives the maximum likelihood estimates

$$\mu = N^{-1}\sum_{n=1}^{N} x_n,$$

and

$$\Sigma = N^{-1}\sum_{n=1}^{N}(x_n-\mu)\,(x_n-\mu)^T.$$

Although this example was not included in Dwyer and Macphail's original paper [4], it is included in Dwyer's later paper [3]. It is a similar application concerning the Likelihood Ratio Method in computational finance [6] which motivated the present author's investigation into this subject.

Returning to Dwyer and Macphail's original paper [4], it is interesting to note the notation they used to express their results, and the correspondence to the results presented in this paper. Using $\langle\cdot\rangle_{i,j}$ to denote the $(i,j)^{th}$ element of a matrix, and defining

$J_{i,j}$ and $K_{i,j}$ to be matrices which are zero apart from a unit value for the $(i,j)^{th}$ element, then their equivalent of the equations for the matrix inverse are

$$\begin{aligned} \frac{\partial A^{-1}}{\partial \langle A \rangle_{i,j}} &= -A^{-1}\, J_{i,j}\, A^{-1}, \\ \frac{\partial \langle A^{-1} \rangle_{i,j}}{\partial A} &= -A^{-T}\, K_{i,j}\, A^{-T}. \end{aligned}$$

In the forward mode, defining the input scalar to be $S_I = A_{i,j}$ for a particular choice (i,j) gives $\dot{A} = J_{i,j}$ and hence, in our notation with $B = A^{-1}$,

$$\dot{B} = -A^{-1} \dot{A}\, A^{-1}.$$

Similarly, in reverse mode, defining the output scalar to be $S_O = (A^{-1})_{i,j}$ for a particular choice (i,j) gives $\overline{B} = K_{i,j}$ and so

$$\overline{A} = -A^{-T}\, \overline{B}\, A^{-T},$$

again matching the result derived previously.

3 Eigenvalues and singular values

3.1 Eigenvalues and eigenvectors

Suppose that A is a square matrix with distinct eigenvalues. We define D to be the diagonal matrix of eigenvalues d_k, and U to be the matrix whose columns are the corresponding eigenvectors U_k, so that $A\,U = U\,D$. The matrices D and U are the quantities returned by the MATLAB function `eig` and the objective in this section is to determine their forward and reverse mode sensitivities.

Differentiation gives

$$\mathrm{d}A\; U + A\; \mathrm{d}U = \mathrm{d}U\; D + U\; \mathrm{d}D.$$

Defining matrix $\mathrm{d}C = U^{-1}\,\mathrm{d}U$ so that $\mathrm{d}U = U\;\mathrm{d}C$, then

$$\mathrm{d}A\, U + U\, D\, \mathrm{d}C = U\, \mathrm{d}C\, D + U\, \mathrm{d}D,$$

and pre-multiplying by U^{-1} and re-arranging gives

$$\mathrm{d}C\, D - D\, \mathrm{d}C + \mathrm{d}D = U^{-1} \mathrm{d}A\, U.$$

Using the notation $A \circ B$ to denote the Hadamard product of two matrices of the same size, defined by each element being the product of the corresponding elements of the input matrices, so that $(A \circ B)_{i,j} = A_{i,j} B_{i,j}$, then

$$\mathrm{d}C\, D - D\, \mathrm{d}C = E \circ \mathrm{d}C$$

where $E_{i,j} = d_j - d_i$. Since the diagonal elements of this are zero, it follows that

$$\mathrm{d}D = I \circ (U^{-1}\mathrm{d}A\,U).$$

The off-diagonal elements of $\mathrm{d}C$ are given by the off-diagonal elements of the equation

$$E \circ \mathrm{d}C + \mathrm{d}D = U^{-1}\mathrm{d}A\,U.$$

The diagonal elements depend on the choice of normalisation for the eigenvectors. Usually, they are chosen to have unit magnitude, but if the subsequent use of the eigenvectors is unaffected by their magnitude it is more convenient to set the diagonal elements of $\mathrm{d}C$ to zero and so

$$\mathrm{d}C = F \circ (U^{-1}\mathrm{d}A\,U), \quad \Longrightarrow \quad \mathrm{d}U = U\left(F \circ (U^{-1}\mathrm{d}A\;U)\right),$$

where $F_{i,j} = (d_j - d_i)^{-1}$ for $i \neq j$, and zero otherwise. Hence, the forward mode sensitivity equations are

$$\begin{aligned} \dot{D} &= I \circ (U^{-1}\dot{A}\,U), \\ \dot{U} &= U\left(F \circ (U^{-1}\dot{A}\;U)\right). \end{aligned}$$

In reverse mode, using the identity $\mathrm{Tr}(A\,(B \circ C)) = \mathrm{Tr}((A \circ B^T)\,C)$, we get

$$\begin{aligned} \mathrm{Tr}\left(\overline{D}^T\mathrm{d}D + \overline{U}^T\mathrm{d}U\right) &= \mathrm{Tr}\left(\overline{D}^T U^{-1}\mathrm{d}A\;U\right) + \mathrm{Tr}\left(\overline{U}^T U\left(F \circ (U^{-1}\mathrm{d}A\;U)\right)\right) \\ &= \mathrm{Tr}\left(\overline{D}^T U^{-1}\mathrm{d}A\;U\right) + \mathrm{Tr}\left(\left((\overline{U}^T U) \circ F^T\right)\;U^{-1}\mathrm{d}A\;U\right) \\ &= \mathrm{Tr}\left(U\left(\overline{D}^T + (\overline{U}^T U) \circ F^T\right)\;U^{-1}\mathrm{d}A\right) \end{aligned}$$

and so

$$\overline{A} = U^{-T}\left(\overline{D} + F \circ (U^T\,\overline{U})\right)\;U^T.$$

3.2 Singular value decomposition

The SVD decomposition of a matrix A of dimension $m \times n$ is

$$A = U\,S\,V^T$$

where S has the same dimensions as A and has zero entries apart from the main diagonal which has non-negative real values arranged in descending order. U and V are square orthogonal real matrices of dimension m and n, respectively. U, S and V are the quantities returned by the MATLAB function `svd` and the objective is to determine their forward and reverse mode sensitivities.

Differentiation gives

$$\mathrm{d}A = \mathrm{d}U\,S\,V^T + U\,\mathrm{d}S\,V^T + U\,S\,\mathrm{d}V^T.$$

Defining matrices $\mathrm{d}C = U^{-1}\,\mathrm{d}U$ and $\mathrm{d}D = V^{-1}\,\mathrm{d}V$ so that $\mathrm{d}U = U\,\mathrm{d}C$ and $\mathrm{d}V = V\,\mathrm{d}D$, then

$$\mathrm{d}A = U\,\mathrm{d}C\,S\,V^T + U\,\mathrm{d}S\,V^T + U\,S\,\mathrm{d}D^T\,V^T,$$

and pre-multiplying by U^T and post-multiplying by V then gives

$$U^T\mathrm{d}A\,V = \mathrm{d}C\,S + \mathrm{d}S + S\,\mathrm{d}D^T. \tag{3.1}$$

Now since $U^TU = I$, differentiation gives

$$\mathrm{d}U^TU + U^T\mathrm{d}U = 0 \quad \Longrightarrow \mathrm{d}C^T + \mathrm{d}C = 0,$$

and similarly $\mathrm{d}D^T + \mathrm{d}D = 0$ as well. Thus, $\mathrm{d}C$ and $\mathrm{d}D$ are both anti-symmetric and have zero diagonals. It follows that

$$\mathrm{d}S = I \circ (U^T\mathrm{d}A\,V),$$

where I is a rectangular matrix of dimension $m \times n$, with unit values along the main diagonal, and zero elsewhere.

In forward mode, this gives

$$\dot{S} = I \circ (U^T\dot{A}\,V).$$

In reverse mode, if we assume the output scalar depends only on the singular values S and not on U and V, so that $\overline{U} = 0$ and $\overline{V} = 0$, then

$$\begin{aligned}
\mathrm{Tr}(\,\overline{S}^T\mathrm{d}S) &= \mathrm{Tr}\left(\,\overline{S}^T\left(I \circ (U^T\mathrm{d}A\,V)\right)\right) \\
&= \mathrm{Tr}\left((\overline{S}^T \circ I^T)(U^T\mathrm{d}A\,V)\right) \\
&= \mathrm{Tr}(\,\overline{S}^TU^T\mathrm{d}A\;V) \\
&= \mathrm{Tr}(V\,\overline{S}^TU^T\mathrm{d}A),
\end{aligned}$$

and hence

$$\overline{A} = U\,\overline{S}\;V^T.$$

To determine $\mathrm{d}U$ and $\mathrm{d}V$, it will be assumed that the singular values are distinct, and that $m \le n$ (if $m > n$ then one can consider the SVD of A^T). Let S, $\mathrm{d}S$ and $\mathrm{d}D$ be partitioned as follows:

$$S = \left(\;S_1 \;\middle|\; 0\;\right), \qquad \mathrm{d}S = \left(\;\mathrm{d}S_1 \;\middle|\; 0\;\right), \qquad \mathrm{d}D = \left(\begin{array}{c|c} \mathrm{d}D_1 & -\mathrm{d}D_2 \\ \hline \mathrm{d}D_2^T & \mathrm{d}D_3 \end{array}\right),$$

where S_1, $\mathrm{d}S_1$ and $\mathrm{d}D_1$ all have dimensions $m \times m$. Furthermore, let $U^T\mathrm{d}A\,V$ be partitioned to give

$$U^T\mathrm{d}A\,V = \left(\;\mathrm{d}P_1 \;\middle|\; \mathrm{d}P_2\;\right).$$

Remembering that $\mathrm{d}D_1$ is antisymmetric, Equation (3.1) then splits into two pieces,

$$\begin{aligned} \mathrm{d}P_1 &= \mathrm{d}C\, S_1 + \mathrm{d}S_1 - S_1\, \mathrm{d}D_1, \\ \mathrm{d}P_2 &= S_1\, \mathrm{d}D_2. \end{aligned}$$

The second of these can be solved immediately to get

$$\mathrm{d}D_2 = S_1^{-1} \mathrm{d}P_2.$$

To solve the other equation, we first take its transpose, giving

$$\mathrm{d}P_1^T = -S_1\, \mathrm{d}C + \mathrm{d}D_1 S_1.$$

It then follows that

$$\begin{aligned} \mathrm{d}P_1\, S_1 + S_1\, \mathrm{d}P_1^T &= \mathrm{d}C\, S_1^2 - S_1^2\, \mathrm{d}C \\ S_1\, \mathrm{d}P_1 + \mathrm{d}P_1^T\, S_1 &= \mathrm{d}D_1 S_1^2 - S_1^2\, \mathrm{d}D_1. \end{aligned}$$

Hence,

$$\begin{aligned} \mathrm{d}C &= F \circ (\mathrm{d}P_1\, S_1 + S_1\, \mathrm{d}P_1^T) \\ \mathrm{d}D_1 &= F \circ (S_1\, \mathrm{d}P_1 + \mathrm{d}P_1^T\, S_1), \end{aligned}$$

where $F_{i,j} = (s_j^2 - s_i^2)^{-1}$ for $i \neq j$, and zero otherwise. Note that these solutions for $\mathrm{d}C$ and $\mathrm{d}D_1$ are antisymmetric because of the antisymmetry of F.

Finally, the value of $\mathrm{d}D_3$ is unconstrained apart from the fact that it must be antisymmetric. The simplest choice is to set it to zero. $\mathrm{d}U$ and $\mathrm{d}V$ can then be determined from $\mathrm{d}C$ and $\mathrm{d}D$, and the reverse mode value for $\overline{A}$ could also be determined from these and the expression for $\mathrm{d}S$.

4 Cholesky factorisation

Given a symmetric positive definite matrix A of dimension N, the Cholesky factorisation determines the lower-triangular matrix L such that $A = LL^T$. There are many uses for a Cholesky factorisation, but one important application is the generation of correlated Normally distributed random numbers [6]. If x is a random vector whose elements are independent Normal variables with zero mean and unit variance, then $y = Lx$ is a vector whose elements are Normal with zero mean and covariance $A = LL^T$.

Pseudo-code for the calculation of L is as follows:

```
for i from 1 to N
   for j from 1 to i
      for k from 1 to j−1
         A_ij := A_ij − L_ik L_jk
      end
      if j=i
         L_ii := √A_ii
      else
         L_ij := A_ij / L_jj
      endif
   end
end
```

The corresponding pseudo-code for calculating $\dot{L}$ is

```
for i from 1 to N
   for j from 1 to i
      for k from 1 to j−1
         Ȧ_ij := Ȧ_ij − L̇_ik L_jk − L_ik L̇_jk
      end
      if j=i
         L̇_ii := ½ Ȧ_ii / L_ii
      else
         L̇_ij := (Ȧ_ij − L_ij L̇_jj) / L_jj
      endif
   end
end
```

and the adjoint code for the calculation of $\overline{A}$, given $\overline{L}$, is

```
for i from N to 1
   for j from i to 1
      if j=i
         Ā_ii := ½ L̄_ii / L_ii
      else
         Ā_ij := L̄_ij / L_jj
         L̄_jj := L̄_jj − L̄_ij L_ij / L_jj
      endif
      for k from j−1 to 1
         L̄_ik := L̄_ik − Ā_ij L_jk
         L̄_jk := L̄_jk − Ā_ij L_ik
      end
   end
end
```

5 Matrix norms

5.1 Frobenius norm

The Frobenius norm of matrix A is defined as

$$B = \|A\|_F = \sqrt{\mathrm{Tr}(A^T A)}.$$

Differentiating this gives

$$\mathrm{d}B = (2\,B)^{-1}\,\mathrm{Tr}(\mathrm{d}A^T A + A^T\,\mathrm{d}A) = B^{-1}\,\mathrm{Tr}(A^T\,\mathrm{d}A),$$

since $\mathrm{Tr}(\mathrm{d}A^T A) = \mathrm{Tr}(A^T\,\mathrm{d}A)$. Thus, in forward mode we have

$$\dot{B} = B^{-1}\,\mathrm{Tr}(A^T \dot{A}),$$

while in reverse mode

$$\overline{B}\,\mathrm{d}B = \mathrm{Tr}(\,\overline{B}\,B^{-1}A^T\mathrm{d}A)$$

and hence

$$\overline{A} = \overline{B}\,B^{-1}A.$$

5.2 Spectral norm

The spectral norm, or 2-norm, of matrix A

$$B = \|A\|_2,$$

is equal to its largest singular value. Hence, using the results from the singular value section, in forward mode we have

$$\dot{B} = U_1^T \dot{A} V_1,$$

where U_1 and V_1 are the first columns of the SVD orthogonal matrices U and V, while in reverse mode

$$\overline{A} = \overline{B}\,U_1\,V_1^T.$$

6 Validation

All results in this paper have been validated with a MATLAB code, given in the appendix, which performs two checks.

The first check uses a wonderfully simple technique based on the Taylor series expansion of an analytic function of a complex variable [15]. If $f(x)$ is analytic with respect to each component of x, and $y{=}f(x)$ is real when x is real, then

$$\dot{y} = \lim_{\varepsilon\to 0}\ \mathcal{I}\{\varepsilon^{-1} f(x{+}\mathrm{i}\varepsilon\dot{x})\}.$$

Taking $\varepsilon = 10^{-20}$ this is used to check the forward mode derivatives to machine accuracy. Note that this is similar to the use of finite differences, but without roundoff inaccuracy.

The requirement that $f(x)$ be analytic can require some creativity in applying the check. For example, the singular values of a complex matrix are always real, and so they cannot be an analytic function of the input matrix. However, for real matrices, the singular values are equal to the square root of the eigenvalues of $A^T A$, and these eigenvalues are an analytic function of A.

The second check is that when inputs A, B lead to an output C, then the identity

$$\mathrm{Tr}(\overline{C}^T \dot{C}) = \mathrm{Tr}(\overline{A}^T \dot{A}) + \mathrm{Tr}(\overline{B}^T \dot{B}),$$

should be satisfied for all $\dot{A}, \dot{B}$ and $\overline{C}$. This check is performed with randomly chosen values for these matrices.

7 Conclusions

This paper has reviewed a number of matrix derivative results in numerical linear algebra. These are useful in applying both forward and reverse mode algorithmic differentiation at a higher level than the usual binary instruction level considered by most AD tools. As well as being helpful for applications which use numerical libraries to perform certain computationally intensive tasks, such as solving a system of simultaneous equations, it could be particularly relevant to those programming in MATLAB or developing AD tools for MATLAB [1, 2, 5, 18].

Acknowledgements

I am grateful to Shaun Forth for the Kubota reference, Andreas Griewank for the Minka and Magnus & Neudecker references, and Nick Trefethen for the Mathai and Stewart & Sun references.

This research was funded in part by a research grant from Microsoft Corporation, and in part by a fellowship from the UK Engineering and Physical Sciences Research Council.

References

[1] C.H Bischof, H.M. Bücker, B. Lang, A. Rasch, and A. Vehreschild. Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In *Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2002)*, pages 65–72. IEEE Computer Society, 2002.

[2] T.F. Coleman and A. Verma. ADMIT-1: Automatic differentiation and MATLAB interface toolbox. *ACM Transactions on Mathematical Software*, 26(1):150–175, 2000.

[3] P.S. Dwyer. Some applications of matrix derivatives in multivariate analysis. *Journal of the American Statistical Association*, 62(318):607–625, 1967.

[4] P.S. Dwyer and M.S. Macphail. Symbolic matrix derivatives. *The Annals of Mathematical Statistics*, 19(4):517–534, 1948.

[5] S.A. Forth. An efficient overloaded implementation of forward mode automatic differentiation in MATLAB. *ACM Transactions on Mathematical Software*, 32(2):195–222, 2006.

[6] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer-Verlag, New York, 2004.

[7] A. Griewank. *Evaluating derivatives : principles and techniques of algorithmic differentiation*. SIAM, 2000.

[8] N.J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1193, 2005.

[9] K. Kubota. Matrix inversion algorithms by means of automatic differentiation. *Applied Mathematics Letters*, 7(4):19–22, 1994.

[10] J.R. Magnus and H. Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons, 1988.

[11] A.M. Mathai. *Jacobians of matrix transformations and functions of matrix argument*. World Scientific, New York, 1997.

[12] T.P. Minka. Old and new matrix algebra useful for statistics. `http://research.microsoft.com/~minka/papers/matrix/`, 2000.

[13] C.R. Rao. *Linear statistical inference and its applications*. Wiley, New York, 1973.

[14] G.S. Rogers. *Matrix derivatives*. Marcel Dekker, New York, 1980.

[15] W. Squire and G. Trapp. Using complex variables to estimate derivatives of real functions. *SIAM Review*, 10(1):110–112, 1998.

[16] M.S. Srivastava and C.G. Khatri. *An introduction to multivariate statistics*. North Holland, New York, 1979.

[17] G.W. Stewart and J. Sun. *Matrix perturbation theory*. Academic Press, 1990.

[18] A. Verma. ADMAT: automatic differentiation in MATLAB using object oriented methods. In *SIAM Interdiscplinary Workshop on Object Oriented Methods for Interoperability*, pages 174–183. SIAM, 1998.

Appendix A MATLAB validation code

```
%
% test code to check results in paper
%

function test

%
% create random test matrices
%

N = 10;

randn('state',0);

% the next line ensures the eigenvalues of A
% are all real, which is needed for the CVT check

A  = 0.1*randn(N) + diag(1:N);
B  = randn(N);
I  = eye(N);

dA = randn(N);
dB = randn(N);
bC = randn(N);

eps  = 1e-20;
epsi = 1/eps;

Ae = A + i*eps*dA;
Be = B + i*eps*dB;

%
% addition
%

Ce = Ae + Be;
C  = real(Ce) ;

dC = dA + dB;

bA = bC;
bB = bC;

disp(sprintf('\naddition'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)+dp(dB,bB)-dp(dC,bC)))

%
% multiplication
%

Ce = Ae*Be;
C  = real(Ce) ;
```

```
dC = dA*B + A*dB;

bA = bC*B';
bB = A'*bC;

disp(sprintf('multiplication'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)+dp(dB,bB)-dp(dC,bC)))

%
% inverse
%

Ce = inv(Ae);
C  = real(Ce) ;

dC = - C*dA*C;

bA = -C'*bC*C';
bB = 0*bC;

disp(sprintf('inverse'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)+dp(dB,bB)-dp(dC,bC)))

%
% determinant
%

de = det(Ae);
d  = real(de) ;

dd = d*trace(A\dA);

bd = 1;
bA = bd*d*inv(A');

disp(sprintf('determinant'))
disp(sprintf('CVT error: %g',norm(dd-epsi*imag(de))))
disp(sprintf('adj error: %g\n',dp(dA,bA)-dd*bd))

%
% matrix polynomial
%

a = [1 2 3 4 5];

C = {};

Ce   = a(5)*I;
C{5} = real(Ce);
dC   = 0;

for n = 4:-1:1
  dC = dA*C{n+1} + A*dC;
```

```
  Ce = Ae*Ce + a(n)*I;
  C{n} = real(Ce);
end

bC2 = bC;
bA  = 0;

for n = 1:4
  bA  = bA + bC2*C{n+1}';
  bC2 = A'*bC2;
end

disp(sprintf('matrix polynomial'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)-dp(dC,bC)))

%
% inverse product
%

Ce = Ae\Be;
C  = real(Ce);
dC = A\(dB-dA*C);

bB = (A')\bC;
bA = -bB*C';

disp(sprintf('inverse product'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)+dp(dB,bB)-dp(dC,bC)))

%
% first quadratic form
%

Ce = Be.'*Ae*Be;
C  = real(Ce) ;

dC = dB'*A*B + B'*dA*B + B'*A*dB;

bA = B*bC*B';
bB = A'*B*bC + A*B*bC';

disp(sprintf('first quadratic form'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)+dp(dB,bB)-dp(dC,bC)))

%
% second quadratic form
%

Ce = Be.'*(Ae\Be);
C  = real(Ce) ;

dC = dB'*(A\B) - B'*(A\dA)*(A\B) + B'*(A\dB);
```

```
bA = -(A'\B)*bC*(A\B)';
bB =  (A'\B)*bC + (A\B)*bC';

disp(sprintf('second quadratic form'))
disp(sprintf('CVT error: %g',norm(dC-epsi*imag(Ce))))
disp(sprintf('adj error: %g\n',dp(dA,bA)+dp(dB,bB)-dp(dC,bC)))

%
% eigenvalues and eigenvectors
%

[Ue,De] = eig(Ae);
U  = real(Ue);
D  = real(De);

% next line makes sure diag(C)=0 in notes
Ue = Ue*diag(1./diag(U\Ue));

D  = diag(D);
E  = ones(N,1)*D'- D*ones(1,N);
F  = 1./(E+I) - I;

P  = U\(dA*U);
dD = I.*P;
dU = U * (F.*P);

bD = diag(randn(N,1));
bU = randn(N);

bD = bD + F.*(U'*bU);
bA = U'\(bD*U');

disp(sprintf('eigenvalues and eigenvectors'))
disp(sprintf('CVT error: %g',norm(dD-epsi*imag(De))))
disp(sprintf('CVT error: %g',norm(dU-epsi*imag(Ue))))
disp(sprintf('adj error: %g\n',dp(dA,bA)-dp(dD,bD)-dp(dU,bU)))

%
% singular values
%

[U,S,V] = svd(A);

S = diag(S);

De = eig(Ae.'*Ae);
De = sort(De,1,'descend');
D  = real(De);

dS = diag( I.*(U'*dA*V) );

bS = randn(N,1);
bA = U*diag(bS)*V';

disp(sprintf('singular value'))
disp(sprintf('svd error: %g',norm(S-sqrt(D))))
```

```
disp(sprintf('CVT error: %g',norm(2*S.*dS-epsi*imag(De))))
disp(sprintf('adj error: %g\n',dp(dA,bA)-dp(dS,bS)))

%
% Cholesky factorisation
%

A = A*A';
A = A + i*eps*dA;
dA_sav = dA;

L  = zeros(N);
dL = zeros(N);

for m = 1:N
  for n = 1:m
    for k = 1:n-1
      A(m,n)  = A(m,n)  - L(m,k)*L(n,k);
      dA(m,n) = dA(m,n) - dL(m,k)*L(n,k) - L(m,k)*dL(n,k);
    end

    if m==n
      L(m,m)  = sqrt(A(m,m));
      dL(m,m) = 0.5*dA(m,m)/L(m,m);
    else
      L(m,n)  = A(m,n)/L(n,n);
      dL(m,n) = (dA(m,n)-L(m,n)*dL(n,n))/L(n,n);
    end
  end
end

bL = randn(N);
bL_sav = bL;
bA = zeros(N);

for m = N:-1:1
  for n = m:-1:1
    if m==n
      bA(m,m) = 0.5*bL(m,m)/L(m,m);
    else
      bA(m,n) = bL(m,n)/L(n,n);
      bL(n,n) = bL(n,n) - bL(m,n)*L(m,n)/L(n,n);
    end

    for k = n-1:-1:1
      bL(m,k) = bL(m,k) - bA(m,n)*L(n,k);
      bL(n,k) = bL(n,k) - bA(m,n)*L(m,k);
    end
  end
end

dL = real(dL);
bA = real(bA);
bL = bL_sav;
dA = dA_sav;
```

```
disp(sprintf('Cholesky factorisation'))
disp(sprintf('CVT error: %g',norm(dL-epsi*imag(L))))
disp(sprintf('adj error: %g\n',dp(dA,bA)-dp(dL,bL)))

%
% matrix norms
%

b2 = norm(A,'fro');

be = sqrt( sum(sum(Ae.*Ae)));
b  = real(be);

db = trace(A'*dA) / b;

bb = 1;
bA = (bb/b) * A;

disp(sprintf('matrix Frobenius norm'))
disp(sprintf('norm error: %g',b-b2))
disp(sprintf('CVT error: %g',db-epsi*imag(be)))
disp(sprintf('adj error: %g\n',dp(dA,bA)-db))

b2 = norm(A,2);

[Ue,ee] = eig(Ae.'*Ae);
[ee,j] = max(diag(ee));
be = sqrt(ee);
b  = real(be);

[U,S,V] = svd(A);
b3 = S(1,1);
U1 = U(:,1);
V1 = V(:,1);
db = U1'*dA*V1;

bb = 1;
bA = bb*U1*V1';

disp(sprintf('matrix 2-norm'))
disp(sprintf('norm error: %g',b-b2))
disp(sprintf('norm error: %g',b-b3))
disp(sprintf('CVT error: %g',db-epsi*imag(be)))
disp(sprintf('adj error: %g\n',dp(dA,bA)-db))


%
% dot product function
%

function p = dp(dA,bA)

p = sum(sum(dA.*bA));
```

On my system the MATLAB code produced the following results, but because the errors are due to machine roundoff error they may be different on other systems.

```
addition
CVT error: 7.59771e-16
adj error: 1.77636e-15

multiplication
CVT error: 8.0406e-15
adj error: -7.10543e-15

inverse
CVT error: 3.94176e-16
adj error: 4.44089e-16

determinant
CVT error: 9.31323e-10
adj error: -2.56114e-09

matrix polynomial
CVT error: 1.5843e-11
adj error: -2.18279e-11

inverse product
CVT error: 1.41363e-15
adj error: -2.33147e-15

first quadratic form
CVT error: 3.3635e-14
adj error: -1.7053e-13

second quadratic form
CVT error: 4.8655e-15
adj error: 7.10543e-15

eigenvalues and eigenvectors
CVT error: 1.12743e-13
CVT error: 4.95477e-13
adj error: -6.66134e-16

singular value
svd error: 1.30233e-14
CVT error: 1.04554e-12
adj error: 8.32667e-16

Cholesky factorisation
CVT error: 3.22419e-16
adj error: -7.77156e-16

matrix Frobenius norm
norm error: -3.55271e-15
CVT error: -2.22045e-16
adj error: 0

matrix 2-norm
norm error: -5.32907e-15
norm error: -1.77636e-15
CVT error: 2.22045e-14
adj error: -2.22045e-16
```