Algorithms and Applications on Vector and Parallel Computers
H.J.J. te Riele, Th.J. Dekker and H.A. van der Vorst (Editors)
© Elsevier Science Publishers B.V. (North-Holland), 1987

Downdating the Cholesky factorization[1]

A. Bojanczyk, R. Brent
Centre for Mathematical Analysis
The Australian National University
GPO Box 4, Canberra ACT 2601, Australia

P. van Dooren
Philips Research Laboratory
Avenue Van Becelaere 2, Box 8
B-1170 Brussels, Belgium

F. de Hoog
Division of Mathematics and Statistics
CSIRO, GPO Box 1965
Canberra ACT 2601, Australia

We analyse and compare three algorithms for "downdating" the Cholesky factorization of a positive definite matrix. Although the algorithms are closely related, their numerical properties differ. Two algorithms are stable in a certain "mixed" sense while the other is unstable. As well as comparing the numerical properties of the algorithms, we compare their computational complexity and their suitability for implementation on parallel or vector computers.

1. Introduction

The Cholesky downdating problem considered in this note is: given an upper triangular matrix $R \in \Re^{n \times n}$ and a vector $x \in \Re^n$ such that $R^T R - xx^T$ is positive definite, find an upper triangular matrix U such that

$$U^T U = R^T R - xx^T.$$

By our assumption of positive definiteness, U exists and is unique up to the signs of its diagonal elements (Golub and van Loan [10]).

The Cholesky downdating problem is closely related to that of downdating a QR-factorization. To show this, let

$$A = \begin{bmatrix} x^T \\ \tilde{A} \end{bmatrix} = Q \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where $A \in \Re^{m \times n}$, $m > n$ and $Q \in \Re^{m \times m}$ is an orthogonal matrix. The problem is

1. This work was done while the third author was visiting the Centre for Mathematical Analysis at The Australian National University

to find an orthogonal matrix $\tilde{Q} \in \mathbb{R}^{(m-1)\times(m-1)}$ such that

$$\tilde{A} = \tilde{Q}\begin{bmatrix} U \\ 0 \end{bmatrix}.$$

Thus we have

$$R^T R = A^T A = xx^T + \tilde{A}^T\tilde{A} = xx^T + U^T U$$

which is precisely the Cholesky downdating problem described previously.

The problem of downdating a QR-factorization occurs for example when an observation (such as an outlier) is deleted from a regression. An algorithm for computing $\tilde{Q}$ and U has been described by GOLUB [9] and GILL, GOLUB, MURRAY and SAUNDERS [8] and it is not difficult to show that the algorithm is backward stable in the classical sense (WILKINSON [9]). This rather satisfactory state of affairs is due to the fact that we know precisely which row (in our case, the first row x^T) of A is to be deleted. However, this is not the case when downdating the Cholesky factorization. Here x^T may bear no relation to rows of R and the only requirement is that the matrix $R^T R - xx^T$ is positive definite. Thus, the best we can expect when computations are performed with finite precision is that the computed U is the exact Cholesky factor of $\tilde{R}^T\tilde{R} - \tilde{x}\tilde{x}^T$ where $\tilde{R}$ and $\tilde{x}$ are close to R and x respectively. However, STEWART [13] has shown that under certain circumstances, small perturbations in R and x may lead to large perturbations in U. Thus, we do not recommend that algorithms based on downdating Cholesky factors (as described in the present note) be used to downdate the triangular factor in the QR decomposition.

Nevertheless, the Cholesky downdating problem is important in its own right. Let B be a positive definite matrix for which a Cholesky factorization has been obtained, that is

$$B = R^T R.$$

Now suppose we wish to compute the Cholesky factors of the positive definite matrix

$$\tilde{B} := B - xx^T =: U^T U.$$

Then,

$$U^T U = R^T R - xx^T.$$

This occurs quite often as a subproblem when a positive definite matrix is perturbed by a matrix of low rank. Often the problem can be solved by a sequence of rank-1 updates followed by a sequence of rank-1 downdates. Such a situation occurs for example in structural problems when elements are added and deleted from a design (see for example ARGYRIS and ROY [1]). For another application of Cholesky downdating see BOJANCZYK, BRENT and DE HOOG [3].

In Section 2a, we describe a method implemented in the LINPACK package, "Algorithm A", analysed by STEWART [13]. In Section 2b we describe a

recursive method, "Algorithm B", which has some computational advantages over algorithm A but does not have comparable stability properties. The recursive algorithm is then modified in Section 2c so that its stability properties are improved. The resulting algorithm will be called "Algorithm C".

The main results of this note are given in Section 3, where we present an error analysis of Algorithms B and C. In particular, we show that Algorithm C is not backward stable in the classical sense but does satisfy a "mixed" error bound which shows that it gives forward errors of the same order as an algorithm which is backward stable in the classical sense (see, e.g., DE JONG [4]). A similar result was obtained by STEWART [13] for Algorithm A. The best error bounds that could be obtained for Algorithm B depend on the data and can be arbitrarily large, from which we conclude that it is numerically unstable [4].

In Section 4 we show that Algorithms B and C are preferable to Algorithm A from the point of view of computational complexity, as they require about 20 percent fewer floating point multiplications $(2n^2 + O(n)$ versus $5n^2/2 + O(n))$. We also show that Algorithms B and C are more readily implemented on parallel or vector computers than is Algorithm A. Finally, in Section 5 some numerical results verifying our stability analysis are presented.

To summarize, Algorithm C appears the best overall, it is faster than Algorithm A and more stable than Algorithm B.

2. THREE DIFFERENT APPROACHES

In this section we compare and relate three different approaches that can be used for solving the downdating problem.

2.1. *The LINPACK method (Algorithm A)*

This method is the one implemented in the LINPACK package and is described by STEWART in [13]. A sequence of $(n+1)\times(n+1)$ Givens rotations of the form

$$J_k = \begin{array}{c} \left[\begin{array}{cccccccc} \cos\theta_k & 0 & \cdots & \sin\theta_k & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ -\sin\theta_k & 0 & \cdots & \cos\theta_k & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \leftarrow (k+1)st \text{ row} \\ \\ \\ \\ \end{array} \\ \begin{array}{c} \uparrow \\ (k+1)st \text{ col} \end{array} \end{array}$$

with $k = 1, \cdots, n$, are used to compute U via the relationship

$$J_1 \cdots J_n \begin{bmatrix} 0^T \\ R \end{bmatrix} = \begin{bmatrix} x^T \\ U \end{bmatrix}. \qquad (1)$$

and since $J_1, \cdots, J_n$ are orthogonal matrices, it is easy to verify that

$$R^T R = xx^T + U^T U.$$

Although the right hand side of (1) will always be an upper Hessenberg matrix, we clearly require some restrictions on $J_1, \cdots, J_n$ to ensure that the first row is indeed the required x^T. Let

$$J_n^T \cdots J_1^T e_1 = q =: \begin{bmatrix} \alpha \\ a \end{bmatrix} \qquad (2)$$

where $e_1 = (1, 0, \cdots, 0)^T$. Then, from (1),

$$a^T R = e_1^T J_1 \cdots J_n \begin{bmatrix} 0^T \\ R \end{bmatrix} = x^T.$$

Thus we must have

$$R^T a = x \qquad (3)$$

and

$$\alpha^2 = 1 - a^T a. \qquad (4)$$

We note that a necessary and sufficient condition for $R^T R - xx^T$ to be positive definite is $\alpha^2 > 0$.

The basic steps in the algorithm are therefore as follows. First the n-vector a is obtained by forward substitution from (3) and α is calculated from (4) (the sign of α is immaterial and the positive root of (4) is usually taken). Givens rotations $J_1, \cdots, J_n$ satisfying

$$J_1 \cdots J_n q = e_1$$

are constructed and U is then calculated from (1).

The above algorithm has been analysed by STEWART [13] and his error analysis is summarized in Section 3. Roughly speaking, the algorithm performs as well as can be expected given the potentially ill-conditioned nature of the downdating problem. However, as discussed in Section 4, the algorithm is not particularly well suited to parallel implementation, the main problem being that the forward substitution (3) needs to be completed before the calculation of (1) can commence.

2.2. *A recursive method (Algorithm B)*

We now describe another standard algorithm for downdating which goes back to GOLUB [9]. Although we derive the method from the previous algorithm to show the close connection between them, the usual derivation is based on completely different ideas.

We begin rewriting (1) as

$$\begin{bmatrix} 0^T \\ R \end{bmatrix} = J_n^T \cdots J_1^T \begin{bmatrix} x^T \\ U \end{bmatrix} \qquad (5)$$

and define

$$\begin{bmatrix} x^{(k)T} \\ R^{(k)} \end{bmatrix} := J_k^T \cdots J_1^T \begin{bmatrix} x^T \\ U \end{bmatrix}, \quad k = 0, 1, \cdots, n \qquad (6)$$

where we have used the convention that $J_k^T \cdots J_1^T$ is the identity when $k=0$. It is easy to verify that $x^{(0)} = x$, $R^{(0)} = U$, $x^{(n)} = 0$ and $R^{(n)} = R$. Furthermore, the first k components of $x^{(k)}$ are zeros and the first k rows of $R^{(k)}$ and R are the same as are the last $n-k$ rows of $R^{(k)}$ and U. From (5) and (6),

$$\begin{bmatrix} x^{(k)T} \\ R^{(k)} \end{bmatrix} = J_k^T \begin{bmatrix} x^{(k-1)T} \\ R^{(k-1)} \end{bmatrix}, \quad k = 1, \cdots, n \qquad (7)$$

which can be shown to be equivalent to ([2]):

$$\begin{bmatrix} x^{(k)T} \\ R^{(k-1)} \end{bmatrix} = S_k \begin{bmatrix} x^{(k-1)T} \\ R^{(k)} \end{bmatrix}, \quad k = 1, \cdots, n \qquad (8)$$

where

$$S_k = \begin{array}{c|c} \begin{bmatrix} \sec\theta_k & 0 & \cdots & -\tan\theta_k & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ -\tan\theta_k & 0 & \cdots & \sec\theta_k & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & & \cdot & \cdot & & \cdot \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} & \begin{array}{l} \\ \\ \\ \\ \leftarrow (k+1)st \text{ row} \\ \\ \\ \\ \end{array} \\ \begin{array}{ccccccc} & & & \uparrow & & & \\ & & & (k+1)st \text{ col} & & & \end{array} & \end{array}$$

From (2), $\cos^2\theta_k > \alpha^2$, $k = 1, \cdots, n$ and hence S_k, $k = 1, \cdots, n$ are well defined.

To see that (8) forms the basis of a recursive algorithm to compute U, let us focus attention on the first and $(k+1)$-st row of (8). As noted previously, the k-th row of $R^{(k)}$ is the same as the k-th row of R and is therefore known. Let us suppose that in the k-th step we also know $x^{(k-1)}$. The role played by θ_k in S_k (or equivalently J_k) is to ensure that the k-th component of $x^{(k)}$ is zero. This can be achieved by setting

$$\cos\theta_k = \sqrt{r_{kk}^2 - (x_k^{(k-1)})^2} / r_{kk} \qquad (9)$$

and

$$\sin\theta_k = x_k^{(k-1)} / r_{kk}. \qquad (10)$$

We now calculate $x^{(k)}$ and the k-th row of U (or equivalently the k-th row of $R^{(k-1)}$) from (8) as

$$x_j^{(k)} = (x_j^{(k-1)} - r_{kj}\sin\theta_k)/\cos\theta_k, \quad j=k+1, \cdots, n \tag{11}$$

and

$$u_{kk} = \sqrt{r_{kk}^2 - (x_k^{(k-1)})^2} \tag{12}$$

$$u_{kj} = (-x_j^{(k-1)}\sin\theta_k + r_{kj})/\cos\theta_k, \quad j=k+1, \cdots n. \tag{13}$$

Since $x^{(0)} = x$ is known, the relations (9-13) yield a recursive method for calculating U.

Using (8), it can be shown that

$$\begin{bmatrix} 0^T \\ U \end{bmatrix} = S_n \cdots S_1 \begin{bmatrix} x^T \\ R \end{bmatrix}$$

and it easy to verify that S_k, $k=1, \cdots, n$ are Σ-unitary with

$$\Sigma = \begin{bmatrix} -1 & 0^T \\ 0 & I_n \end{bmatrix},$$

where a matrix A is Σ-unitary if

$$A^T\Sigma A = \Sigma.$$

The product

$$S := S_n \cdots S_1$$

is also Σ-unitary and hence

$$U^T U = [0|U^T]\Sigma \begin{bmatrix} 0^T \\ U \end{bmatrix} = [x|R^T]S^T\Sigma S \begin{bmatrix} x^T \\ R \end{bmatrix}$$

which is an independent verification that U is the required Cholesky factor. Thus, the method can be regarded as that of finding a product of planar Σ-unitary transformations to reduce $[x|R^T]^T$ to $[0|U^T]^T$ in the same way that Givens rotations are used to reduce the matrix in the updating problem (see for example GOLUB [9] and GILL, GOLUB, MURRAY and SAUNDERS [8]).

The recursive method has a number of advantages over Algorithm A. Since there is no forward substitution phase, the operation count is somewhat less. In addition the algorithm is quite well suited to parallel implementation. However, the error analysis in Section 3 indicates that the stability properties of the method are substantially inferior to those of Algorithm A.

2.3. *A modified recursive algorithm (Algorithm C)*

It turns out that Algorithm B can be modified so that its stability properties are substantially improved.

At the k-th step, consider the $(k+1)$-st row of (7). In component form, this can be written as

$$u_{kk} = \sqrt{r_{kk}^2 - (x_k^{(k-1)})^2} \tag{14}$$

$$u_{kj} = (-x_j^{(k-1)}\sin\theta_k + r_{kj})/\cos\theta_k, \quad j=k+1, \cdots, n \tag{15}$$

where $\cos\theta_k$ and $\sin\theta_k$ have been calculated using (9) and (10) respectively. Now, consider the first row of (7) and rewrite it as

$$x_j^{(k)} = \cos\theta_k x_j^{(k-1)} - \sin\theta_k u_{kj}, \quad j=k+1, \cdots, n. \tag{16}$$

This is clearly a well defined calculation as u_{kj}, $j=k+1, \cdots, n$ have already been calculated via (15).

Thus, (9) and (15-16) form the basis of an algorithm for calculating U which differs only slightly from Algorithm B. However, as we shall see in Section 3, this small modification enables us to establish stability estimates that are comparable to those for Algorithm A. In addition, we still retain all of the desirable features of Algorithm B. Note that this method was also suggested in LAWSON and HANSON [12] but without any comments on its stability.

3. ERROR ANALYSIS

We denote quantities stored in the computer with a tilde. In addition, ϵ is the relative precision of the machine considered and terms of order ϵ^2 are neglected.

3.2. *Algorithm A*

An error analysis of this method is to be found in [13]. It is shown there that there exists an exactly orthogonal matrix Q (which is not computed) such that

$$Q\begin{bmatrix} 0^T \\ R \end{bmatrix} = \begin{bmatrix} x^T + \Delta x^T \\ \tilde{U} + \Delta\tilde{U} \end{bmatrix} \tag{17}$$

where

$$\|[\Delta\tilde{U}]_i\|_2 \leqslant 6n\epsilon\|[R]_i\|_2 \tag{18}$$

$$|\Delta x_i| \leqslant [(13n+5)/2 + (i+2)\sqrt{i}]\epsilon\|[R]_i\|_2.$$

Here we use $[\cdot]_i$ to denote the i-th column of a matrix. From this, one easily obtains a bound for the total error matrix.

$$\left\| \begin{bmatrix} \Delta x^T \\ \Delta\tilde{U} \end{bmatrix} \right\|_F \leqslant [n^2/2 + 9n\sqrt{n} + O(n)]\epsilon\|R\|_F. \tag{20}$$

Notice that errors are not only superimposed on the data — R and x in this case — but also on the result $\tilde{U}$. Hence, the bound (20) does not guarantee forward or backward stability of the algorithm, but rather proves what could be called "mixed" stability. As argued in [13], the forward part of the error, $\Delta\tilde{U}$, is in fact unimportant since the "true" forward error $(\tilde{U} - U)$ is usually much larger and mainly depends on Δx, the backward part of the error in (17). The result of mixed stability is thus as satisfactory in practice as backward stability since error bounds in both cases are comparable.

3.2. *Algorithm B*

The error analysis of this method depends on how it is implemented. We therefore write the order of computations for one step, which, without loss of generality, can be the first step as given in (9-13).

Let us denote the elements of the rows involved in this step as follows,

$$1/c_1\begin{bmatrix}1 & -s_1\\ -s_1 & 1\end{bmatrix}\begin{bmatrix}x_1^{(0)} & x_2^{(0)} & \cdots & x_n^{(0)}\\ r_{11} & r_{12} & \cdots & r_{1n}\end{bmatrix}=\begin{bmatrix}0 & x_2^{(1)} & \cdots & x_n^{(1)}\\ u_{11} & u_{12} & \cdots & u_{1n}\end{bmatrix}. \tag{21}$$

Now the operations are performed in the following order (where we use the $fl(\cdot)$ notation of WILKINSON [14]),

$$\begin{aligned}\tilde{u}_{11} &= fl(\sqrt{(r_{11}-x_1^{(0)})(r_{11}+x_1^{(0)})})\\ \tilde{c}_1 &= fl(\tilde{u}_{11}/r_{11})\\ \tilde{s}_1 &= fl(x_1^{(0)}/r_{11})\\ \text{for}\quad i &= 2 \text{ until } n\\ \tilde{u}_{1i} &= fl((r_{1i}-\tilde{s}_1x_i^{(0)})/\tilde{c}_1)\\ \tilde{x}_i^{(1)} &= fl((x_i^{(0)}-\tilde{s}_1r_{1i})/\tilde{c}_1).\end{aligned} \tag{22}$$

In the Appendix we show that for this implementation of (21), the following equality holds,

$$1/c_1\begin{bmatrix}1 & -s_1\\ -s_1 & 1\end{bmatrix}\begin{bmatrix}x_i^{(0)}+\Delta x_i^{(0)}\\ r_{1i}^{(1)}+\Delta r_{1i}\end{bmatrix}=\begin{bmatrix}\tilde{x}_i^{(1)}\\ \tilde{u}_{1i}\end{bmatrix} \tag{23}$$

where

$$\left\|\begin{bmatrix}\Delta x_i^{(0)}\\ \Delta r_{1i}\end{bmatrix}\right\|_2\leqslant(8\epsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2. \tag{24}$$

If we rewrite (23) as

$$1/c_1\begin{bmatrix}1 & -s_1\\ -s_1 & 1\end{bmatrix}\begin{bmatrix}x_i^{(0)}+\Delta x_i^{(0)}\\ r_{1i}\end{bmatrix}=\begin{bmatrix}\tilde{x}_i^{(1)}\\ \tilde{u}_{1i}+\Delta\tilde{u}_{1i}\end{bmatrix} \tag{25}$$

then for the mixed error one obtains (see the Appendix),

$$\left\|\begin{bmatrix}\Delta x_i^{(0)}\\ \Delta\tilde{u}_{1i}\end{bmatrix}\right\|_2\leqslant(8(1+|s_1|)\epsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|^2. \tag{26}$$

Since this 2-vector is now a subcolumn of the i-th column of the matrix $\begin{bmatrix}x^T\\ \tilde{U}\end{bmatrix}$, we have

$$J_1^T\begin{bmatrix}x^T+\Delta^{(1)}x^T\\ \tilde{U}+\Delta^{(1)}\tilde{U}\end{bmatrix}=\begin{bmatrix}\tilde{x}^{(1)T}\\ R^{(1)}\end{bmatrix} \tag{27}$$

with J_1 exactly orthogonal and

$$\left\|\begin{bmatrix}\Delta^{(1)}x^T\\ \Delta^{(1)}\tilde{U}\end{bmatrix}_i\right\|_2\leqslant(8(1+|s_1|)\epsilon/c_1|)\,\|[R]_i\|_2$$

since the i-the column of R and $[x, U^T]^T$ have the same norms.

Errors in the subsequent steps are similarly bounded in terms of the deflated problem $[\tilde{x}^{(1)}, R^{(1)T}]^T$. Because corresponding columns in (27) are related by an orthogonal transformation, the errors of step 2 can be mapped backwards onto $[x, \tilde{U}^T]^T$ without altering their norms. (This would not have been possible if we had used backward errors as in (23), (24)). This now gives

$$J_2^TJ_1^T\begin{bmatrix}x^T+\Delta^{(1)}x^T+\Delta^{(2)}x^T\\ \tilde{U}+\Delta^{(1)}\tilde{U}+\Delta^{(2)}\tilde{U}\end{bmatrix}=\begin{bmatrix}\tilde{x}^{(2)T}\\ R^{(2)}\end{bmatrix} \tag{29}$$

where

$$\begin{aligned}&\left\|\begin{bmatrix}\Delta^{(1)}x^T+\Delta^{(2)}x^T\\ \Delta^{(1)}\tilde{U}+\Delta^{(2)}\tilde{U}\end{bmatrix}_i\right\|_2\\ &\leqslant 8\{(1+|s_1|)/|c_1|+(1+|s_2|)/|c_2|\}\epsilon\|[R]_i\|_2\end{aligned} \tag{30}$$

for all columns i except the first one (since that one is not affected anymore). By induction we finally have

$$J_n^T\cdots J_1^T\begin{bmatrix}x^T+\Delta x^T\\ \tilde{U}+\Delta\tilde{U}\end{bmatrix}=\begin{bmatrix}0^T\\ R\end{bmatrix} \tag{31}$$

where $\Delta\tilde{U}=\sum_{i=1}^n\Delta^{(i)}\tilde{U}$ and $\Delta x=\sum_{i=1}^n\Delta^{(i)}x$ are bounded columnwise by

$$\left\|\begin{bmatrix}\Delta x^T\\ \Delta\tilde{U}\end{bmatrix}_i\right\|_2\leqslant 8i\max_{1\leqslant j\leqslant i}\{(1+|s_j|)/|c_j|\}\epsilon\|[R]_i\|_2 \tag{32}$$

which is significantly worse than (18-19) and problem dependent. For the total error we obtain

$$\left\|\begin{bmatrix}\Delta x^T\\ \Delta\tilde{U}\end{bmatrix}\right\|_F\leqslant 5n\sqrt{n}\max_{1\leqslant j\leqslant n}\{(1+|s_j|)/|c_j|\}\epsilon\|R\|_2 \tag{33}$$

3.3. *Algorithm C*

Since the method is essentially a rearrangement of the previous method, one might expect similar bounds for the numerical errors, but this is not the case because operations are performed in a different order.

Let us again denote the elements of the rows involved in step 1 as

$$\begin{bmatrix}c_1 & -s_1\\ s_1 & c_1\end{bmatrix}\begin{bmatrix}x_1^{(0)}x_2^{(0)} & \cdots & x_n^{(0)}\\ u_{11}u_{12} & \cdots & u_{1n}\end{bmatrix}=\begin{bmatrix}0\ x_2^{(1)} & \cdots & x_n^{(1)}\\ r_{11}\ r_{12} & \cdots & r_{1n}\end{bmatrix} \tag{34}$$

then c_1, s_1 and u_{1j}, $x_j^{(i)}$ are constructed as follows

$$\begin{aligned}
\tilde{u}_{11} &= fl(\sqrt{(r_{11}-x_1^{(0)})(r_{11}+x_1^{(0)})}) \\
\bar{c}_1 &= fl(\tilde{u}_{11}/r_{11}) \\
\bar{s}_1 &= fl(x_1^{(0)}/r_{11}) \\
\text{for} \quad & i = 2 \text{ until } n \qquad\qquad (35)\\
\tilde{u}_{1i} &= fl((r_{1i}-\bar{s}_1 x_i^{(0)})/\bar{c}_1) \\
\bar{x}_i^{(1)} &= fl(\bar{c}_1 x_i^{(0)} - \bar{s}_1 \tilde{u}_{1i}).
\end{aligned}$$

he only difference between this algorithm and the previous one is in the com-utation of $\bar{x}_i^{(1)}$. This single difference allows us to obtain the following bound r the mixed error vector (see the Appendix)

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta \tilde{u}_{1i} \end{bmatrix} \right\|_2 \leq 6.5\epsilon \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \qquad (36)$$

sing this we find by induction that

$$\left\| \begin{bmatrix} \Delta x^T \\ \Delta \tilde{U} \end{bmatrix}_i \right\|_2 \leq 6.5 i\epsilon \| [R]_i \|_2 \qquad (37)$$

nd for the total error matrix

$$\left\| \begin{bmatrix} \Delta x^T \\ \Delta \tilde{U} \end{bmatrix} \right\|_F \leq 4n\sqrt{n}\,\epsilon \|R\|_F \qquad (38)$$

hich compares favourably with the two previous methods (see (20) and (33)).

COMPLEXITY AND PARALLELISM

Iere we compare the computational efficiency of the different methods and omment on the possibility of implementing them in parallel.

1. *The LINPACK method*

The method first constructs the vector q which has length $n+1$ and is given by 2-4). This construction is a forward substitution process for the triangular sys-em (3) and requires $n^2/2 + O(n)$ multiplications. Next the elementary rota-ions J_i, $i = n, n-1, \cdots, 1$, are applied to q and the matrix $[0, R^T]^T$. Because f the triangular shape of R, this requires $2n^2 + O(n)$ multiplications giving a otal operation count of

$$2.5n^2 + O(n) \qquad (39)$$

multiplications. This operation count can be reduced by using modified Givens otations instead of plane rotations, see [6].

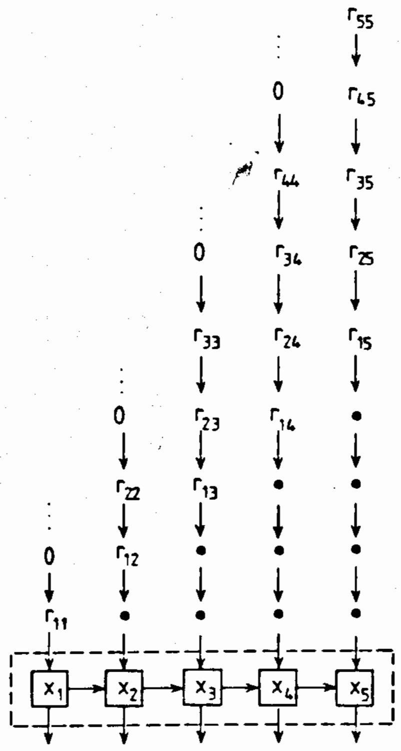

FIGURE 1. A linear systolic array for downdating a Cholesky factor

On systolic-like arrays, both stages of the algorithm can easily be parallelized. The first stage, the back substitution, in $2n$ parallel steps using n processors as depicted in Figure 1. At the beginning of the algorithm each processor contains one element x_i of the row vector x^T, while after the 'passage' of the triangular factor R, they will contain the components a_i of the solution a^T of $x^T = a^T R$ (see e.g. [11]). The computation of α is performed while a is being constructed and thus requires only one additional step for the square root. The second stage requires $2n$ parallel steps with n processors, again using a scheme as depicted in Figure 1. Now the x_i in this figure are in fact the components a_i of the vector q and the rotations J_i are generated as the factor R passes through it and is being transformed to the downdated array U (see e.g. [3]). In total, n processors and $4n$ parallel steps are required here. Notice that apparently the computation of α separates both stages in the method and prevents any concurrency between both stages. But a closer analysis shows that in fact α only provides a test for the existence of the solution and is not really eeded for the construction of the J_i. Both steps could thus be merged on an rray of the type of Figure 1 and the number of time steps would then be alved.

On vector machines all these operations are also easy to exploit. The triangular system (3) can be solved via typical $A \cdot X(\cdot) + Y(\cdot)$ vector operations nd the application of the rotations J_i ammounts to so-called "pipelined ivens" (see e.g. [5]). Both of these operations are known to perform well on ector machines [5].

2. *The recursive Algorithms B and C*

Here in each step one essentially constructs an elementary rotation J_k (or quivalently S_k) and applies it to determine $x^{(k)}$ and the k-th row of U from $^{(k-1)}$ and the k-th row of R. This is easily seen to require $4(n-i+1)$ multiplications for each transformation J_i or S_i, resulting in an operation count of

$$2n^2 + 0(n) \tag{40}$$

nultiplications for the whole process.

Parallel implementation of these methods would require n processors and $2n$ teps as shown above for the second stage of the LINPACK method. Since, ssentially, the first stage is skipped here, these algorithms compare favourably vith the LINPACK method for both systolic arrays and vector machines. Notice also that by using modified elementary rotations [6] or modified Σ-orthogonal transformations, the work could be reduced and square roots could e avoided, just as in the second stage of the previous method.

3. *Consecutive up- and downdates*

As explained in the introduction, it is often required to perform a series of onsecutive up- and/or downdates of Cholesky factorizations. In this case a ifferent scheme than the one suggested in Figure 1 is recommended. Indeed, he Σ-orthogonal method of Algorithm C is completely analogous to the rthogonal method used in recursive updating of Cholesky factors (see [7]). The array of processors shown in Figure 2 used for updating in GENTLEMAN nd KUNG [7] can thus be used also for downdating.

Here the factor R resides now in the $n(n+1)/2$ processors shown in Figure and x^T is "passed through" it leaving the updated factor behind him in the riangular array (see [7] for details). When connecting a "flag" (say ± 1) to ach row x^T in order to indicate if an updating or downdating is requested hen the programs of the processors are easily adapted to take this into ccount and to "switch" from one type of operations to another. In this ashion consecutive up- and/or downdates can easily be pipelined through the riangular array depicted in Figure 2. This in fact constitutes a more efficient se of the processors than in Figure 1, since there the left most processors gradually become inactive as the time steps proceed. In the second scheme, shown n Figure 2, all processors are active at any time step, for as long as new rows ^{T} are fed into the array.

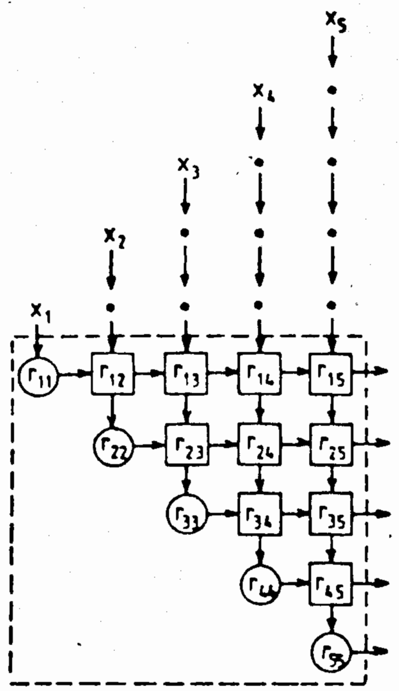

FIGURE 2. A triangular systolic array for up- and downdating a Cholesky factor

5. NUMERICAL RESULTS

The results of STEWART [13] on the conditioning of the downdating problem show that we cannot expect that $\tilde{U}$, the calculated factor, will be close to U. However a mixed stability result of the form

$$R^T R - (x + \Delta x)(x + \Delta x)^T = (\tilde{U} + \Delta\tilde{U})(\tilde{U} + \Delta\tilde{U})^T$$

where Δx and $\Delta\tilde{U}$ are small does ensure that

$$R^T R - xx^T = \tilde{U}\tilde{U}^T + E$$

where E is small. To demonstrate the superior stability properties of Algorithms A and C over Algorithm B, we have computed the quantity

$$e = \frac{\|R^T R - xx^T - \tilde{U}\tilde{U}^T\|_F}{\|U^T U\|_F}$$

for the problem

$$R = \begin{bmatrix} 1 & \sin(\theta/2) \\ 0 & \sqrt{2}\cos(\theta/2) \end{bmatrix}, \quad x = \begin{bmatrix} \sin\theta \\ \cos(\theta/2) \end{bmatrix},$$

hich has the solution

$$U = \begin{bmatrix} \cos\theta & -\sin(\theta/2) \\ 0 & \cos(\theta/2) \end{bmatrix}.$$

esults obtained on a MacIntosh which has working accuracy of between 7 nd 8 significant digits are tabulated in Table 1 for $\cos\theta = 2^{-k}$, $k = 3,6,9,12$.

$\cos\theta$	*Alg A*	*Alg B*	*Alg C*
2^{-3}	$1.337E-8$	$2.367E-7$	$1.183E-7$
2^{-6}	$1.210E-7$	$1.073E-6$	$6.939E-8$
2^{-9}	$1.788E-7$	$3.218E-5$	$2.946E-8$
2^{-12}	$1.264E-7$	$1.011E-4$	$2.467E-8$

TABLE 1.

he numerical results are consistent with the stability analysis of Section 3 and learly demonstrate the superior properties of Algorithms A and C. Moreover, hese results also show that Algorithm B is unstable (in de Jong's sense [4]) hen $\cos\theta$ becomes very small.

PPENDIX

lere we derive the bounds (24), (26) and (36) for the numerical errors incurred n (22) and (35).

Let us denote by δ_i and ϵ_i quantities that are smaller in absolute value than the relative precision of the machine used. Then we have, according to WIL-INSON [14],

$$\begin{aligned} \tilde{u}_{11} &= \sqrt{\{[(r_{11}-x_1^{(0)})(1+\delta_1)(r_{11}+x_1^{(0)})(1+\delta_2)](1+\delta_3)\}}\,(1+\delta_4) = \\ &= u_{11}(1+2.5\epsilon_1) \qquad (41) \\ \tilde{c}_1 &= (\tilde{u}_{11}/r_{11})(1+\delta_5) = (u_{11}/r_{11})(1+3.5\epsilon_2) = c_1(1+3.5\epsilon_2) \qquad (42) \\ \tilde{s}_1 &= (x_1^{(0)}/r_{11})(1+\delta_6) = (x_1^{(0)}/r_{11})(1+\epsilon_3) = s_1(1+\epsilon_3) \qquad (43) \end{aligned}$$

nd for each i

$$\begin{aligned} \tilde{x}_i^{(1)} &= [x_i^{(0)} - \tilde{s}_1 r_{1i}(1+\delta_{10})](1+\delta_{11}) / [\tilde{c}_1(1+\delta_{12})] \\ &= [x_i^{(0)} - \tilde{s}_1 r_{1i}(1+\epsilon_6)] / [\tilde{c}_1(1+2\epsilon_7)] \qquad (44) \\ \tilde{u}_{1i} &= [r_{1i} - \tilde{s}_1 x_i^{(0)}(1+\delta_7)](1+\delta_8) / [\tilde{c}_1(1+\delta_9)] \\ &= [r_{1i} - \tilde{s}_1 x_i^{(0)}(1+\epsilon_4)] / [\tilde{c}_1(1+2\epsilon_5)]. \qquad (45) \end{aligned}$$

Multiplying these with their respective denominators yields

$$\begin{aligned} x_i^{(0)} &= \tilde{s}_1 r_{1i}(1+\epsilon_6) + \tilde{x}_i^{(1)} \tilde{c}_1(1+2\epsilon_7) \\ r_{1i} &= \tilde{s}_i x_i^{(0)}(1+\epsilon_4) + \tilde{u}_{1i}\tilde{c}_1(1+2\epsilon_5) \end{aligned} \qquad (46)$$

Using (42) and (43) we finally obtain

$$\begin{aligned} x_i^{(0)} - s_1 r_{1i} &= c_1 \tilde{x}_i^{(1)} + 2\epsilon_8 s_1 r_{1i} + 5.5\epsilon_9 \tilde{x}_i^{(1)} c_1 \\ r_{1i} - s_1 x_i^{(0)} &= c_1 \tilde{u}_{1i} + 2\epsilon_{10} s_1 x_i^{(0)} + 5.5\epsilon_{11} c_1 \tilde{u}_{1i} \end{aligned} \qquad (47)$$

or

$$1/c_1 \begin{bmatrix} 1 & -s_1 \\ -s_1 & 1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} \\ r_{1i} \end{bmatrix} = \begin{bmatrix} \tilde{x}_i + \Delta\tilde{x}_i^{(1)} \\ \tilde{u}_{1i} + \Delta\tilde{u}_{1i} \end{bmatrix}$$

$$= \begin{bmatrix} \tilde{x}_i^{(1)} + 2\epsilon_8(s_1/c_1) r_{1i} + 5.5\epsilon_9 \tilde{x}_i^{(1)} \\ \tilde{u}_{1i} + 2\epsilon_{10}(s_1/c_1) x_i^{(0)} + 5.5\epsilon_{11}\tilde{u}_{1i} \end{bmatrix}. \qquad (48)$$

Identifying the error terms in the right hand side with the vector $[\Delta\tilde{x}_i^{(1)}, \Delta\tilde{u}_{1i}]^T$ we then obtain from the Cauchy-Schwartz inequality the bounds

$$\begin{aligned} |\Delta\tilde{x}_i^{(1)}| &\leqslant (1/|c_1|) \left\| \begin{bmatrix} 5.5\epsilon_9 \tilde{x}_i^{(1)} \\ 2\epsilon_8 r_{1i} \end{bmatrix} \right\|_2 \leqslant (5.5\epsilon/|c_1|) \left\| \begin{bmatrix} \tilde{x}_i^{(1)} \\ r_{1i} \end{bmatrix} \right\|_2 \\ &= (5.5\epsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2 + O(\epsilon^2) \\ |\Delta\tilde{u}_{1i}| &\leqslant (1/|c_1|) \left\| \begin{bmatrix} 2\epsilon_{10} x_i^{(0)} \\ 5.5\epsilon_{11}\tilde{u}_{1i} \end{bmatrix} \right\|_2 \leqslant (5.5\epsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2 \end{aligned} \qquad (49)$$

and subsequently

$$\begin{aligned} \left\| \begin{bmatrix} \Delta\tilde{x}_i^{(1)} \\ \Delta\tilde{u}_{1i} \end{bmatrix} \right\|_2 &\leqslant (5.5\sqrt{2}\epsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2 \\ &\leqslant (8\epsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \end{aligned} \qquad (50)$$

For the backward and mixed errors one obtains via similar techniques the following bounds

$$\left\| \begin{bmatrix} \Delta\tilde{x}_i^{(1)} \\ \Delta r_{1i} \end{bmatrix} \right\|_2 \leqslant (8\epsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2 \qquad (51)$$

$$\left\| \begin{bmatrix} \Delta\tilde{x}_i^{(1)} \\ \Delta\tilde{u}_{1i} \end{bmatrix} \right\|_2 \leqslant (8\epsilon(1+|s_1|)/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \qquad (52)$$

Notice that all three bounds depend on $1/|c_1|$ which is large when the condition number κ of S_1,

$$\kappa(S_1) = (1+|s_1|)/(1-|s_1|) = (1+|s_1|)^2/|c_1|^2 \qquad (53)$$

is large.

Now we consider Algorithm C. A similar analysis to t used for lgorithm B yields

$$\tilde{x}_i^{(1)} = [\tilde{c}_1 x_i^{(0)}(1+\delta_{10}) - \tilde{s}_1 \tilde{u}_{1i}(1+\delta_{11})](1+\delta_{12})$$
$$= \tilde{c}_1 x_i^{(0)}(1+2\epsilon_{12}) - \tilde{s}_1 \tilde{u}_{1i}(1+2\epsilon_{13}). \qquad (54)$$

ogether with the second equation of (46) and using (42), (43), this now leads

$$\begin{bmatrix} \tilde{x}_i^{(1)} \\ r_{1i} \end{bmatrix} = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} + \begin{bmatrix} -3\epsilon_{14} s_1 \tilde{u}_{1i} + 5.5\epsilon_{15} c_1 x_i^{(0)} \\ 5.5\epsilon_{11} c_1 \tilde{u}_{1i} + 2\epsilon_{10} s_1 x_i^{(0)} \end{bmatrix}. \qquad (55)$$

dentifying the error terms on the right hand side with the vector $\Delta\tilde{x}_i^{(1)}, -\Delta r_{1i}]^T$ and using a similar argument as in the proof of (49) and 0), one now finds

$$\left\| \begin{bmatrix} \Delta\tilde{x}_i^{(1)} \\ \Delta r_{1i} \end{bmatrix} \right\|_2 \leq 6.5\epsilon \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \qquad (56)$$

rom the relation

$$\begin{bmatrix} \Delta\tilde{x}_i^{(0)} \\ \Delta\tilde{u}_{1i} \end{bmatrix} = \begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} \Delta\tilde{x}_i^{(1)} \\ \Delta r_{1i} \end{bmatrix} \qquad (57)$$

ne then obtains a similar bound for the vector $[\Delta x_i^{(0)}, \Delta\tilde{u}_{1i}]^T$, which completes he proof.

EFERENCES

J.H. ARGYRIS, J.R. ROY, (1972). *General treatments of structural modifications*. J. Structural Division, A.S.C.E., 98, ST2, Proc. Paper 8732, pp. 465-492.

V. BELEVITCH, (1968). *Classical network theory*. Holden Day, San Francisco.

A.W. BOJANCZYK, R.P. BRENT, F.R. DE HOOG, (May 1985). *QR factorization of Toeplitz matrices*, Report CMA-R07-1985, Centre for Math. Anal., The Australian National University.

L. DE JONG, (1977). *Towards a formal definition of numerical stability*, Numerische Mathematik, Vol. 28, pp. 211-220.

J.J. DONGARRA, D.C. SORENSEN, (1986). *Linear algebra on high performance computers*, Appl. Math. & Comp., Vol. 20, pp. 57-88.

W.M. GENTLEMAN, (1973). *Least squares computations by Givens transformations without square roots*. JIMA, vol. 12, pp. 329-336.

W.M. GENTLEMAN, H.T. KUNG, (1981). *Matrix triangularization by systolic arrays*, Proceedings SPIE Symp. (1981), Vol. 298, Real Time Signal Processing IV, pp. 19-26.

P.E. GILL, G.H. GOLUB, W. MURRAY, M.A. SAUNDERS, (1974). *Methods for modifying matrix factorizations*. Mathematics of Computation, vol. 28, pp. 505-535.

G.H. GOLUB, (1969). ***Matrix decompositions and statistical calculations***, in:

Statistical Computation, Eds. R.C. Milton, J.A. Nelder, pp. 365-397, Academic Press, New York.

10. G.H. GOLUB, C. VAN LOAN, (1983). *Matrix Computations*. The John Hopkins Press, Baltimore.

11. H.T. KUNG, C.E. LEISERSON, (1980). *Algorithms for VLSI processor arrays*, in *Introduction to VLSI systems* (C. Mead and L. Conway, eds.), Addision-Wesley, pp. 271-292.

12. C.L. LAWSON, R.J. HANSON, (1974). *Solving least squares problems*. Prentice Hall, Englewood Cliffs, N.J.

13. G.W. STEWART, (1979). *The effect of rounding errors on an algorithm for downdating a Cholesky factorization*. JIMA, vol. 23, pp. 203-213.

14. J.H. WILKINSON, (1965). *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford.