SIAM J. SCI. STAT. COMPUT.
Vol. 8, No. 3, May 1987

© 1987 Society for Industrial and Applied Mathematics
002

A NOTE ON DOWNDATING THE CHOLESKY FACTORIZATION*

A. W. BOJANCZYK†¶, R. P. BRENT†, P. VAN DOOREN‡ AND F. R. DE HOOG§

Abstract. 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. In addition to comparing the numerical properties of the algorithms, we compare their computational complexity and their suitability for implementation on parallel or vector computers.

Key words. Cholesky factorization, numerical stability, computational complexity, parallel computation, forward error analysis, backward error analysis, mixed stability, downdating, updating, rank-one correction, Sigma-unitary transformation, hyperbolic transformation, orthogonal transformation

AMS(MOS) subject classifications. 65F05, 65F30, 65G05

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

$$\mathbf{U}^T\mathbf{U} = \mathbf{R}^T\mathbf{R} - \mathbf{x}\mathbf{x}^T.$$

By our assumption of positive definiteness, $\mathbf{U}$ exists and is unique up to the signs of its diagonal elements (Golub and van Loan [7]).

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

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

where $\mathbf{A} \in R^{m \times n}$, $m > n$ and $\mathbf{Q} \in R^{m \times m}$ is an orthogonal matrix. The problem is to find an orthogonal matrix $\tilde{\mathbf{Q}} \in R^{(m-1)\times(m-1)}$ such that

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

Thus we have

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

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 $\tilde{\mathbf{Q}}$ and $\mathbf{U}$ has been described by Gill, Golub, Murray and Saunders [5] and it is not difficult to show that the algorithm is backward stable in the classical sense (Wilkinson [10]). This rather satisfactory state of affairs is due to the fact that we know precisely

* Received by the editors September 11, 1985; accepted for publication (in revised form) February 17, 1986.

† Centre for Mathematical Analysis, The Australian National University, GPO Box 4, Canberra ACT 2601, Australia.

‡ Philips Research Laboratory, Av. van Becelaere 2, Box 8, B-1170 Brussels, Belgium.

§ Division of Mathematics and Statistics, CSIRO, GPO Box 1965, Canberra ACT 2601, Australia.

¶ Present address, Department of Computer Science, Washington University, St. Louis, Missouri 63130.

which row (in our case, the first row $\mathbf{x}^T$) of $\mathbf{A}$ is to be deleted. However, this is not the case when downdating the Cholesky factorization. Here $\mathbf{x}^T$ may bear no relation to rows of $\mathbf{R}$ and the only requirement is that the matrix $\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T$ 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 $\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}-\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$ where $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{x}}$ are close to $\mathbf{R}$ and $\mathbf{x}$, respectively. However, Stewart [9] has shown that under certain circumstances, small perturbations in $\mathbf{R}$ and $\mathbf{x}$ 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 $\mathbf{B}$ be a positive definite matrix for which a Cholesky factorization has been obtained, that is

$$\mathbf{B}=\mathbf{R}^T\mathbf{R}.$$

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

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

Then,

$$\mathbf{U}^T\mathbf{U}=\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T.$$

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 § 2a, we describe a method implemented in the LINPACK package, "Algorithm A," analysed by Stewart [9]. In § 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 § 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 § 3, where we present an error analysis of Algorithms B and C. In particular, we show that Algorithm C is not 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. A similar result was obtained by Stewart [9] 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 probably numerically unstable.

In § 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 § 5 some numerical results verifying our stability analysis are presented.

To summarize, Algorithm C appears the best overall, since 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.

2a. The LINPACK method (Algorithm A). This method is the one implemented in the LINPACK package and is described by Stewart in [9]. A sequence of $(n+1)\times(n+1)$ Givens rotations of the form

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

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

$$\mathbf{J}_1\cdots\mathbf{J}_n\begin{bmatrix}\mathbf{O}^T\\ \mathbf{R}\end{bmatrix}=\begin{bmatrix}\mathbf{x}^T\\ \mathbf{U}\end{bmatrix}, \tag{2.1}$$

and since $\mathbf{J}_1,\cdots,\mathbf{J}_n$ are orthogonal matrices, it is easy to verify that

$$\mathbf{R}^T\mathbf{R}=\mathbf{x}\mathbf{x}^T+\mathbf{U}^T\mathbf{U}.$$

Although the right-hand side of (2.1) will always be an upper Hessenberg matrix, we clearly require some restrictions on $\mathbf{J}_1,\cdots,\mathbf{J}_n$ to ensure that the first row is indeed the required $\mathbf{x}^T$. Let

$$\mathbf{J}_n^T\cdots\mathbf{J}_1^T\mathbf{e}_1=\mathbf{q}=:\begin{bmatrix}\alpha\\ \mathbf{a}\end{bmatrix}, \tag{2.2}$$

where $\mathbf{e}_1=(1,0,\cdots,0)^T$. Then, from (2.1),

$$\mathbf{a}^T\mathbf{R}=\mathbf{e}_1^T\mathbf{J}_1\cdots\mathbf{J}_n\begin{bmatrix}\mathbf{O}^T\\ \mathbf{R}\end{bmatrix}=\mathbf{x}^T.$$

Thus we must have

$$\mathbf{R}^T\mathbf{a}=\mathbf{x} \tag{2.3}$$

and

$$\alpha^2=1-\mathbf{a}^T\mathbf{a}. \tag{2.4}$$

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

The basic steps in the algorithm are therefore as follows. First the n-vector $\mathbf{a}$ is obtained by forward substitution from (2.3) and α is calculated from (2.4) (the sign of α is immaterial and the positive root of (2.4) is usually taken). Givens rotations $\mathbf{J}_1,\cdots,\mathbf{J}_n$ satisfying

$$\mathbf{J}_1\cdots\mathbf{J}_n\mathbf{q}=\mathbf{e}_1$$

are constructed and $\mathbf{U}$ is then calculated from (2.1). Note that Algorithm A modifies the rows of $\mathbf{R}$ from bottom to top, whereas Algorithms B and C (described below) process them from top to bottom.

The algorithm has been analysed by Stewart [9] and his error analysis is summarised in § 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 § 4, the algorithm is not particularly well suited to parallel implementation, the main problem being that the forward substitution (2.3) needs to be completed before the calculation of (2.1) can commence.

2b. A recursive method (Algorithm B). We now describe another standard algorithm for downdating which may be found for example in Lawson and Hanson [8]. 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 by rewriting (2.1) as

$$\begin{bmatrix} \mathbf{O}^T \\ \mathbf{R} \end{bmatrix} = \mathbf{J}_n^T \cdots \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T \\ \mathbf{U} \end{bmatrix} \tag{2.5}$$

and define

$$\begin{bmatrix} (\mathbf{x}^{(k)})^T \\ \mathbf{R}^{(k)} \end{bmatrix} := \mathbf{J}_k^T \cdots \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T \\ \mathbf{U} \end{bmatrix}, \qquad k = 0, 1, \cdots, n, \tag{2.6}$$

where we have used the convention that $\mathbf{J}_k^T \cdots \mathbf{J}_1^T$ is the identity when $k = 0$. It is easy to verify that $\mathbf{x}^{(0)} = \mathbf{x}$, $\mathbf{R}^{(0)} = \mathbf{U}$, $\mathbf{x}^{(n)} = \mathbf{O}$ and $\mathbf{R}^{(n)} = \mathbf{R}$. Furthermore, the first k components of $\mathbf{x}^{(k)}$ are zeros and the first k rows of $\mathbf{R}^{(k)}$ and $\mathbf{R}$ are the same as are the last $n-k$ rows of $\mathbf{R}^{(k)}$ and $\mathbf{U}$. From (2.5) and (2.6),

$$\begin{bmatrix} (\mathbf{x}^{(k)})^T \\ \mathbf{R}^{(k)} \end{bmatrix} = \mathbf{J}_k^T \begin{bmatrix} \mathbf{x}(^{(k-1)})^T \\ \mathbf{R}^{(k-1)} \end{bmatrix}, \qquad k = 1, \cdots, n, \tag{2.7}$$

which can be shown to be equivalent to

$$\begin{bmatrix} (\mathbf{x}^{(k)})^T \\ \mathbf{R}^{(k-1)} \end{bmatrix} = \mathbf{S}_k \begin{bmatrix} \mathbf{x}(^{(k-1)})^T \\ \mathbf{R}^{(k)} \end{bmatrix}, \qquad k = 1, \cdots, n, \tag{2.8}$$

where

$$\mathbf{S}_k = \begin{array}{cl} \begin{bmatrix} \sec\Theta_k & 0 & \cdots & 0 & -\tan\Theta_k & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ -\tan\Theta_k & 0 & \cdots & 0 & \sec\Theta_k & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & 1 \end{bmatrix} & \begin{array}{l} \\ \\ \\ \\ \leftarrow (k+1)\text{st row.} \\ \\ \\ \\ \end{array} \\ \begin{array}{cccccccc} & & & & \uparrow & & & \\ & & & & (k+1)\text{st col} & & & \end{array} & \end{array}$$

From (2.2), $\cos^2\Theta_k > \alpha^2$, $k = 1, \cdots, n$ and hence $\mathbf{S}_k$, $k = 1, \cdots, n$ are well defined.

To see that (2.8) forms the basis of a recursive algorithm to compute $\mathbf{U}$, let us focus attention on the first and $(k+1)$st rows of (2.8). As noted previously, the kth row of $\mathbf{R}^{(k)}$ is the same as the kth row of $\mathbf{R}$ and is therefore known. Let us suppose that in the kth step we also know $\mathbf{x}^{(k-1)}$. The role played by Θ_k in $\mathbf{S}_k$ (or equivalently $\mathbf{J}_k$) is to ensure that the kth component of $\mathbf{x}^{(k)}$ is zero. This can be achieved by setting

$$\cos\Theta_k = (r_{kk}^2 - (x_k^{(k-1)})^2)^{1/2}/r_{kk} \tag{2.9a}$$

and

$$\sin\Theta_k = x_k^{(k-1)}/r_{kk}. \tag{2.9b}$$

We now calculate $\mathbf{x}^{(k)}$ and the kth row of $\mathbf{U}$ (or equivalently the kth row of $\mathbf{R}^{(k-1)}$) from (2.8) as

$$x_j^{(k)} = (x_j^{(k-1)} - r_{kj}\sin\Theta_k)/\cos\Theta_k, \qquad j=k+1,\cdots,n \tag{2.10a}$$

and

$$u_{kk} = (r_{kk}^2 - (x_k^{(k-1)})^2)^{1/2} \tag{2.10b}$$

$$u_{kj} = (-x_j^{(k-1)}\sin\Theta_k + r_{kj})/\cos\Theta_k, \qquad j=k+1,\cdots,n. \tag{2.10c}$$

Since $\mathbf{x}^{(0)}=\mathbf{x}$ is known, the relations (2.9), (2.10) yield a recursive method for calculating $\mathbf{U}$.

Using (2.8), it can be shown that

$$\begin{bmatrix}\mathbf{O}^T\\ \mathbf{U}\end{bmatrix} = \mathbf{S}_n\cdots\mathbf{S}_1\begin{bmatrix}\mathbf{x}^T\\ \mathbf{R}\end{bmatrix}$$

and it is easy to verify that $\mathbf{S}_k$, $k=1,\cdots,n$ are Σ-unitary with

$$\Sigma = \begin{bmatrix}-1 & \mathbf{O}^T\\ \mathbf{O} & \mathbf{I}\end{bmatrix},$$

where a matrix $\mathbf{A}$ is Σ-unitary iff

$$\mathbf{A}^T\Sigma\mathbf{A} = \Sigma.$$

The product

$$\mathbf{S} := \mathbf{S}_n\cdots\mathbf{S}_1$$

is also Σ-unitary and hence

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

which is an independent verification that $\mathbf{U}$ is the required Cholesky factor. Thus, the method can be regarded as that of finding a product of planar Σ-unitary transformations to reduce $|\mathbf{x}|\mathbf{R}^T]^T$ to $[\mathbf{O}|\mathbf{U}^T]^T$ in the same way that Givens rotations are used to reduce the matrix in the updating problem (see for, example Gill, Golub, Murray and Saunders [5]).

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 § 3 indicates that the stability properties of the method are substantially inferior to those of Algorithm A.

2c. 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 kth step, consider the $(k+1)$st row of (2.7). In component form, this can be rewritten as

$$u_{kk}=(r_{kk}^2-(x_k^{(k-1)})^2)^{1/2}, \tag{2.11a}$$

$$u_{kj}=(-x_j^{(k-1)}\sin\Theta_k+r_{kj})/\cos\Theta_k, \qquad j=k+1,\cdots,n, \tag{2.11b}$$

where $\cos\Theta_k$ and $\sin\Theta_k$ have been calculated using (2.9a) and (2.9b), respectively. Now, consider the first row of (2.7) and rewrite it as

$$x_j^{(k)}=\cos\Theta_k x_j^{(k-1)}-\sin\Theta_k u_{kj}, \qquad j=k+1,\cdots,n. \tag{2.11c}$$

This is clearly a well defined calculation as u_{kj}, $j=k+1,\cdots,n$ have already been calculated via (2.11b).

Thus, (2.9) and (2.11) form the basis of an algorithm for calculating $\mathbf{U}$ which differs only slightly from Algorithm B. However, as we shall see in § 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.

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.

3a. Algorithm A. An error analysis of this method is to be found in [9]. It is shown there that there exists an exactly orthogonal matrix $\hat{\mathbf{Q}}$ (which is not computed) such that

$$\hat{\mathbf{Q}}\begin{bmatrix}\mathbf{O}^T\\ \mathbf{R}\end{bmatrix}=\begin{bmatrix}\mathbf{x}^T+\Delta\mathbf{x}^T\\ \tilde{\mathbf{U}}+\Delta\tilde{\mathbf{U}}\end{bmatrix}, \tag{3.1}$$

where

$$\|[\Delta\tilde{\mathbf{U}}]_i\|_2\leqq 6n\varepsilon\|[\mathbf{R}]_i\|_2, \tag{3.2}$$

$$|\Delta x_i|\leqq[(13n+5)/2+(i+2)\sqrt{i}]\varepsilon\|[\mathbf{R}]_i\|_2. \tag{3.3}$$

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

$$\left\|\begin{bmatrix}\Delta\mathbf{x}^T\\ \Delta\tilde{\mathbf{U}}\end{bmatrix}\right\|_F\leqq[n^2/2+9n\sqrt{n}+O(n)]\varepsilon\|\mathbf{R}\|_F. \tag{3.4}$$

Notice that errors are not only superimposed on the data—$\mathbf{R}$ and $\mathbf{x}$ in this case—but also on the result $\tilde{\mathbf{U}}$. Hence, the bound (3.4) does not guarantee forward or backward stability of the algorithm, but rather proves what could be called "mixed" stability. As argued in [9], the forward part of the error, $\Delta\tilde{\mathbf{U}}$, is in fact unimportant since the "true" forward error $(\tilde{\mathbf{U}}-\mathbf{U})$ is usually much larger and mainly depends on $\Delta\mathbf{x}$, the backward part of the error in (3.1). The result of mixed stability is thus as satisfactory in practice as backward stability since error bounds in both cases are comparable.

3b. 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 (2.9), (2.10).

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

$$1/c_1\begin{bmatrix}1 & -s_1\\ -s_1 & 1\end{bmatrix}\begin{bmatrix}x_1^{(0)} & x_2^{(0)} & \cdots & x_n^{(0)}\\ r_{11} & r_{12} & \cdots & r_{1n}\end{bmatrix}=\begin{bmatrix}0 & x_2^{(1)} & \cdots & x_n^{(1)}\\ u_{11} & u_{12} & \cdots & u_{1n}\end{bmatrix}. \tag{3.5}$$

Now the operations are performed in the following order (where we use the fl(·) notation of Wilkinson [10]):

$$\begin{aligned} &\tilde{u}_{11} = \mathrm{fl}(\sqrt{(r_{11} - x_1^{(0)})(r_{11} + x_1^{(0)})}),\\ &\tilde{c}_1 = \mathrm{fl}(\tilde{u}_{11}/r_{11}),\\ &\tilde{s}_1 = \mathrm{fl}(x_1^{(0)}/r_{11}),\\ &\tilde{u}_{1i} = \mathrm{fl}((r_{1i} - \tilde{s}_1 x_i^{(0)})/\tilde{c}_1),\\ &\tilde{x}_i^{(1)} = \mathrm{fl}((x_i^{(0)} - \tilde{s}_1 r_{1i})/\tilde{c}_1) \quad \text{for } i = 2, \cdots, n. \end{aligned} \tag{3.6}$$

In the Appendix we show that for this implementation of (3.5), the following equality holds,

$$1/c_1 \begin{bmatrix} 1 & -s_1 \\ -s_1 & 1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} + \Delta x_i^{(0)} \\ r_{1i} + \Delta r_{1i} \end{bmatrix} = \begin{bmatrix} \tilde{x}_1^{(1)} \\ \tilde{u}_{1i} \end{bmatrix}, \tag{3.7}$$

where

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta r_{1i} \end{bmatrix} \right\|_2 \leqq (8\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{3.8}$$

If we rewrite (3.7) as

$$1/c_1 \begin{bmatrix} 1 & -s_1 \\ -s_1 & 1 \end{bmatrix} \begin{bmatrix} x_i^{(0)} + \Delta x_i^{(0)} \\ r_{1i} \end{bmatrix} = \begin{bmatrix} \tilde{x}_i^{(1)} \\ \tilde{u}_{1i} + \Delta \tilde{u}_{1i} \end{bmatrix}, \tag{3.9}$$

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

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta \tilde{u}_{1i} \end{bmatrix} \right\|_2 \leqq (8(1 + |s_1|)\varepsilon/|c_1|) \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{3.10}$$

Since this 2-vector is now a subcolumn of the ith column of the matrix $[\mathbf{x}, \tilde{\mathbf{U}}^T]^T$, we have

$$\mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T + \Delta^{(1)}\mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta^{(1)}\tilde{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} (\tilde{\mathbf{x}}^{(1)})^T \\ \mathbf{R}^{(1)} \end{bmatrix} \tag{3.11}$$

with $\mathbf{J}_1$ exactly orthogonal and

$$\left\| \begin{bmatrix} \Delta^{(1)}\mathbf{x}^T \\ \Delta^{(1)}\tilde{\mathbf{U}} \end{bmatrix}_i \right\|_2 \leqq (8(1 + |s_1|)\varepsilon/|c_1|) \|[\mathbf{R}]_i\|_2, \tag{3.12}$$

since the ith columns of $\mathbf{R}$ and $[\mathbf{x}, \mathbf{U}^T]^T$ have the same norms.

Errors in the subsequent step are similarly bounded in terms of the deflated problem $[\tilde{\mathbf{x}}^{(1)}, (\mathbf{R}^{(1)})^T]^T$. Because corresponding columns in (3.11) are related by an orthogonal transformation, the errors of step 2 can be mapped backwards onto $[\mathbf{x}, \tilde{\mathbf{U}}^T]^T$ without altering their norms. (This would not have been possible if we had used backward errors as in (3.7), (3.8).) This now gives

$$\mathbf{J}_2^T \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T + \Delta^{(1)}\mathbf{x}^T + \Delta^{(2)}\mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta^{(1)}\tilde{\mathbf{U}} + \Delta^{(2)}\tilde{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} (\tilde{\mathbf{x}}^{(2)})^T \\ \mathbf{R}^{(2)} \end{bmatrix}, \tag{3.13}$$

where

$$\left\| \begin{bmatrix} \Delta^{(1)}\mathbf{x}^T + \Delta^{(2)}\mathbf{x}^T \\ \Delta^{(1)}\tilde{\mathbf{U}} + \Delta^{(2)}\tilde{\mathbf{U}} \end{bmatrix}_i \right\|_2 \leqq 8\{(1 + |s_1|)/|c_1| + (1 + |s_2|)/|c_2|\}\varepsilon \|[\mathbf{R}]_i\|_2 \tag{3.14}$$

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

$$\mathbf{J}_n^T \cdots \mathbf{J}_1^T \begin{bmatrix} \mathbf{x}^T + \Delta\mathbf{x}^T \\ \tilde{\mathbf{U}} + \Delta\tilde{\mathbf{U}} \end{bmatrix} = \begin{bmatrix} \mathbf{O}^T \\ \mathbf{R} \end{bmatrix}, \tag{3.15}$$

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

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix}_i \right\|_2 \leqq 8i \max_{1 \leqq j \leqq i} \{(1+|s_j|)/|c_j|\} \varepsilon \|[\mathbf{R}]_i\|_2 \tag{3.16}$$

which is significantly worse than (3.3) and problem dependent. For the total error we obtain

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix} \right\|_F \leqq 5n\sqrt{n} \max_{1 \leqq i \leqq n} \{(1+|s_i|)/|c_i|\} \varepsilon \|\mathbf{R}\|_F. \tag{3.17}$$

3c. Algorithm C. Since this 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

$$\begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} x_1^{(0)} & x_2^{(0)} & \cdots & x_n^{(0)} \\ u_{11} & u_{12} & \cdots & u_{1n} \end{bmatrix} = \begin{bmatrix} 0 & x_2^{(1)} & \cdots & x_n^{(1)} \\ r_{11} & r_{12} & \cdots & r_{1n} \end{bmatrix}; \tag{3.18}$$

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

$$\begin{aligned} &\tilde{u}_{11} = \mathrm{fl}(\sqrt{(r_{11} - x_1^{(0)})(r_{11} + x_1^{(0)})}), \\ &\tilde{c}_1 = \mathrm{fl}(\tilde{u}_{11}/r_{11}), \\ &\tilde{s}_1 = \mathrm{fl}(x_1^{(0)}/r_{11}), \\ &\tilde{u}_{1i} = \mathrm{fl}((r_{1i} - \tilde{s}_1 x_i^{(0)})/\tilde{c}_1), \\ &\tilde{x}_i^{(1)} = \mathrm{fl}(\tilde{c}_1 x_i^{(0)} - \tilde{s}_1 \tilde{u}_{1i}) \quad \text{for } i = 2, \cdots, n. \end{aligned} \tag{3.19}$$

The only difference between this algorithm and the previous one is in the computation of $\tilde{x}_i^{(1)}$. This single difference allows us to obtain the following bound for the mixed error vector (see the Appendix):

$$\left\| \begin{bmatrix} \Delta x_i^{(0)} \\ \Delta\tilde{u}_{1i} \end{bmatrix} \right\|_2 \leqq 6.25\varepsilon \left\| \begin{bmatrix} x_i^{(0)} \\ \tilde{u}_{1i} \end{bmatrix} \right\|_2. \tag{3.20}$$

Using this, we find by induction that

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix} \right\|_2 \leqq 6.25 i \varepsilon \|[\mathbf{R}]_i\|_2 \tag{3.21}$$

and for the total error matrix

$$\left\| \begin{bmatrix} \Delta\mathbf{x}^T \\ \Delta\tilde{\mathbf{U}} \end{bmatrix} \right\|_F \leqq 4n\sqrt{n}\varepsilon \|\mathbf{R}\|_F \tag{3.22}$$

which compares favourably with the two previous methods (see (3.4) and (3.16)).

4. Complexity. Here we compare the computational efficiency of the different methods and comment on the possibility of implementing them in parallel.

4a. The LINPACK method. The method first constructs the vector $\mathbf{q}$ which has length $n+1$ and is given by (2.2) and (2.3). This construction is a forward substitution process for the triangular system (2.2) and requires $n^2/2+O(n)$ multiplications. Next the elementary rotations $\mathbf{J}_i$, $i=n, n-1, \cdots, 1$, are applied to $\mathbf{q}$ and the matrix $[\mathbf{O}, \mathbf{R}^T]^T$. Because of the triangular shape of $\mathbf{R}$, this requires $2n^2+O(n)$ multiplications giving a total operation count of

$$2.5n^2+O(n) \tag{4.1}$$

multiplications. This operation count can be reduced by using modified Givens rotations instead of plane rotations (see [4]), or by using an LDL^T version of the method (see [6]).

Both stages of the algorithm can be parallelized; the first stage, the back substitution, in $2n$ parallel steps using n processors and the second stage in $2(n+1)$ parallel steps using $n+1$ processors. The computation of α is performed while $\mathbf{a}$ is being constructed and thus requires only one additional step for the square root. It is this computation of α that separates both stages in the method and prevents any concurrency between both stages. In total $(n+1)$ processors and $4(n+1)$ parallel steps are required.

4b. The recursive Algorithms B and C. Here in each step one essentially constructs an elementary rotation $\mathbf{J}_k$ (or equivalently $\mathbf{S}_k$) and applies it to determine $\mathbf{x}^{(k)}$ and the kth row of $\mathbf{U}$ from $\mathbf{x}^{(k-1)}$ and the kth row of $\mathbf{R}$. This is easily seen to require $4(n-i+1)$ multiplications for each transformation $\mathbf{J}_i$ or $\mathbf{S}_i$, resulting in an operation count of

$$2n^2+O(n) \tag{4.2}$$

multiplications for the whole process.

Parallel implementation of these methods would require n processors and $2n$ steps which compares favourably with the LINPACK method. Notice also that by using modified elementary rotations [4] or modified Σ-orthogonal transformations, the work could be reduced and square roots could be avoided.

5. Numerical results. The results of Stewart [9] on the conditioning of the downdating problem show that we cannot expect that $\tilde{\mathbf{U}}$, the calculated factor, will be close to $\mathbf{U}$. However a mixed stability result of the form

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

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

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

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

$$\frac{\|\mathbf{R}^T\mathbf{R}-\mathbf{x}\mathbf{x}^T-\tilde{\mathbf{U}}\tilde{\mathbf{U}}^T\|_F}{\|\mathbf{U}^T\mathbf{U}\|_F}$$

for the problem

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

which has the solution

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

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

TABLE 1

$\cos\theta$	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

The numerical results are consistent with the stability analysis of § 3 and clearly demonstrate the superior stability properties of Algorithms A and C.

Appendix. Here we derive the bounds (3.8), (3.10) and (3.20) for the numerical errors incurred in (3.6) and (3.19).

Let us denote by δ_i and ε_i quantities that are smaller in absolute value than ε, the relative precision of the machine used. Then we have, according to Wilkinson [10]:

$$\tilde{u}_{11}=\sqrt{\{[(r_{11}-x_1^{(0)})(1+\delta_1)(r_{11}+x_1^{(0)})(1+\delta_2)](1+\delta_3)\}}(1+\delta_4)=u_{11}(1+2.5\varepsilon_1), \tag{A.1}$$

$$\tilde{c}_1=(\tilde{u}_{11}/r_{11})(1+\delta_5)=(u_{11}/r_{11})(1+3.5\varepsilon_2)=c_1(1+3.5\varepsilon_2), \tag{A.2}$$

$$\tilde{s}_1=(x_1^{(0)}/r_{11})(1+\delta_6)=(x_1^{(0)}/r_{11})(1+\varepsilon_3)=s_1(1+\varepsilon_3), \tag{A.3}$$

and for each i,

$$\begin{aligned}\tilde{x}_i^{(1)}&=[x_i^{(0)}-\tilde{s}_1r_{1i}(1+\delta_{10})](1+\delta_{11})/[\tilde{c}_1(1+\delta_{12})]\\&=[x_i^{(0)}-\tilde{s}_1r_{1i}(1+\varepsilon_6)]/[\tilde{c}_1(1+2\varepsilon_7)],\end{aligned} \tag{A.4}$$

$$\begin{aligned}\tilde{u}_{1i}&=[r_{1i}-\tilde{s}_1x_i^{(0)}(1+\delta_7)](1+\delta_8)/[\tilde{c}_1(1+\delta_9)]\\&=[r_{1i}-\tilde{s}_1x_i^{(0)}(1+\varepsilon_4)]/[\tilde{c}_1(1+2\varepsilon_5)].\end{aligned} \tag{A.5}$$

Multiplying these with their respective denominators yields

$$\begin{aligned}x_i^{(0)}&=\tilde{s}_1r_{1i}(1+\varepsilon_6)+\tilde{x}_i^{(1)}\tilde{c}_1(1+2\varepsilon_7),\\r_{1i}&=\tilde{s}_1x_i^{(0)}(1+\varepsilon_4)+\tilde{u}_{1i}\tilde{c}_1(1+2\varepsilon_5).\end{aligned} \tag{A.6}$$

Using (A.2) and (A.3), we finally obtain

$$\begin{aligned}x_i^{(0)}-s_1r_{1i}&=c_1\tilde{x}_i^{(1)}+2\varepsilon_8s_1r_{1i}+5.5\varepsilon_9\tilde{x}_i^{(1)}c_1,\\r_{1i}-s_1x_i^{(0)}&=c_1\tilde{u}_{1i}+2\varepsilon_{10}s_1x_i^{(0)}+5.5\varepsilon_{11}c_1\tilde{u}_{1i},\end{aligned} \tag{A.7}$$

or

$$1/c_1\begin{bmatrix}1 & -s_1\\ -s_1 & 1\end{bmatrix}\begin{bmatrix}x_i^{(0)}\\ r_{1i}\end{bmatrix}=\begin{bmatrix}\tilde{x}_i^{(1)}+\Delta\tilde{x}_i^{(1)}\\ \tilde{u}_{1i}^{(1)}+\Delta\tilde{u}_{1i}\end{bmatrix}=\begin{bmatrix}\tilde{x}_i^{(1)}+2\varepsilon_8(s_1/c_1)r_{1i}+5.5\varepsilon_9\tilde{x}_i^{(1)}\\ \tilde{u}_{1i}+2\varepsilon_{10}(s_1/c_1)x_i^{(0)}+5.5\varepsilon_{11}\tilde{u}_{1i}\end{bmatrix}. \tag{A.8}$$

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

$$\begin{aligned}|\Delta\tilde{x}_i^{(1)}|&\leqq(1/|c_1|)\left\|\begin{bmatrix}5.5\varepsilon_9\tilde{x}_i^{(1)}\\ 2\varepsilon_8r_{1i}\end{bmatrix}\right\|_2\leqq(5.5\varepsilon/|c_1|)\left\|\begin{bmatrix}\tilde{x}_i^{(1)}\\ r_{1i}\end{bmatrix}\right\|_2\\&=(5.5\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2+O(\varepsilon^2),\\|\Delta\tilde{u}_{1i}|&\leqq(1/|c_1|)\left\|\begin{bmatrix}2\varepsilon_{10}x_i^{(0)}\\ 5.5\varepsilon_{11}\tilde{u}_{1i}\end{bmatrix}\right\|_2\leqq(5.5\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2,\end{aligned} \tag{A.9}$$

and subsequently,

$$\left\|\begin{bmatrix}\Delta\tilde{x}_i^{(1)}\\ \Delta\tilde{u}_{1i}\end{bmatrix}\right\|_2 \leqq (5.5\sqrt{2}\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2 \leqq (8\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2. \tag{A.10}$$

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

$$\left\|\begin{bmatrix}\Delta x_i^{(0)}\\ \Delta r_{1i}\end{bmatrix}\right\|_2 \leqq (8\varepsilon/|c_1|)\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2, \tag{A.11}$$

$$\left\|\begin{bmatrix}\Delta x_i^{(0)}\\ \Delta\tilde{u}_{1i}\end{bmatrix}\right\|_2 \leqq [8\varepsilon(1+|s_1|)/|c_1|]\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2. \tag{A.12}$$

Notice that all three bounds depend on $1/|c_1|$ which is large when the condition number of $\mathbf{S}_1$,

$$\kappa(\mathbf{S}_1)=(1+|s_1|)/(1-|s_1|)=(1+|s_1|)^2/|c_1|^2, \tag{A.13}$$

is large.

Now we consider Algorithm C. A similar analysis to that used for Algorithm B now yields

$$\begin{aligned}\tilde{x}_i^{(1)}&=[c_1x_i^{(0)}(1+\delta_{10})-\tilde{s}_1\tilde{u}_{1i}(1+\delta_{11})](1+\delta_{12})\\ &=\tilde{c}_1x_i^{(0)}(1+2\varepsilon_{12})-\tilde{s}_1\tilde{u}_{1i}(1+2\varepsilon_{13}).\end{aligned} \tag{A.14}$$

Together with the second equation of (A.6) and using (A.2), (A.3), this now leads to

$$\begin{bmatrix}\tilde{x}_i^{(1)}\\ r_{1i}\end{bmatrix}=\begin{bmatrix}c_{11} & -s_1\\ s_1 & c_1\end{bmatrix}\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}+\begin{bmatrix}-3\varepsilon_{14}s_1\tilde{u}_{1i}+5.5\varepsilon_{15}c_1x_i^{(0)}\\ 5.5\varepsilon_{11}c_1\tilde{u}_{1i}+2\varepsilon_{10}s_1x_i^{(0)}\end{bmatrix}. \tag{A.15}$$

Identifying the errors 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 (A.9) and (A.10), one now finds

$$\left\|\begin{bmatrix}\Delta\tilde{x}_i^{(1)}\\ \Delta r_{1i}\end{bmatrix}\right\|_2 \leqq 6.25\varepsilon\left\|\begin{bmatrix}x_i^{(0)}\\ \tilde{u}_{1i}\end{bmatrix}\right\|_2. \tag{A.16}$$

From the relation

$$\begin{bmatrix}\Delta x_i^{(0)}\\ \Delta\tilde{u}_{1i}\end{bmatrix}=\begin{bmatrix}c_1 & s_1\\ -s_1 & c_1\end{bmatrix}\begin{bmatrix}\Delta\tilde{x}_i^{(1)}\\ \Delta r_{1i}\end{bmatrix} \tag{A.17}$$

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

Acknowledgments. We thank Professor G. H. Golub and Dr. M. A. Saunders for their helpful comments on a draft of this paper.

REFERENCES

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

[2] V. BELOVITCH, *Classical Network Theory*, Holden Day, San Francisco, CA, 1968.

[3] A. W. BOJANCZYK, R. P. BRENT AND F. R. DE HOOG, *QR factorization of Toeplitz matrices*, Report CMA-R07-1985, Centre for Math. Anal., The Australian National University, May 1985, Numerische Mathematik, to appear.

[4] M. GENTLEMAN, *Least squares computations by Givens transformations without square roots*, J. Inst. Math. Appl., 12 (1973), pp. 329–336.

[5] P. E. GILL, G. H. GOLUB, W. MURRAY AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[6] P. E. GILL, W. MURRAY AND M. A. SAUNDERS, *Methods for computing and modifying the LDV factors of a matrix*, Math. Comp., 29 (1975), pp. 1051–1077, esp. p. 1063.

[7] G. H. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins Press, Baltimore, MD, 1983.

[8] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[9] G. W. STEWART, *The effect of rounding errors on an algorithm for downdating a Cholesky factorization*, J. Inst. Math. Appl., 23 (1979), pp. 203–213.

[10] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, England, 1965.

[11] G. H. GOLUB, *Matrix decompositions and statistical calculations*, in Statistical Computation, R. C. Multon and J. A. Nelder, eds., Academic Press, New York, 1969, pp. 365–387.