UMIACS-TR-90-86 July 1990
CS-TR 2494 Revised January 1991

An Updating Algorithm for Subspace Tracking

G. W. Stewart*

ABSTRACT

In certain signal processing applications it is required to compute the null space of a matrix whose rows are samples of a signal with p components. The usual tool for doing this is the singular value decomposition. However, the singular value decomposition has the drawback that it requires $O(p^3)$ operations to recompute when a new sample arrives. In this paper, we show that a different decomposition, called the URV, decomposition is equally effective in exhibiting the null space and can be updated in $O(p^2)$ time. The updating technique can be run on a linear array of p processors in $O(p)$ time.

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. This work was supported in part by the Air Force Office of Scientific Research under Contract AFOSR-87-0188.

An Updating Algorithm for Subspace Tracking

G. W. Stewart*

ABSTRACT

In certain signal processing applications it is required to compute the null space of a matrix whose rows are samples of a signal with p components. The usual tool for doing this is the singular value decomposition. However, the singular value decomposition has the drawback that it requires $O(p^3)$ operations to recompute when a new sample arrives. In this paper, we show that a different decomposition, called the URV, decomposition is equally effective in exhibiting the null space and can be updated in $O(p^2)$ time. The updating technique can be run on a linear array of p processors in $O(p)$ time.

1. Introduction

Many problems in digital signal processing require the computation of an approximate null space of an $n \times p$ matrix A whose rows represent samples of a signal (see [9] for examples and references). Specifically, we must find an orthogonal matrix $V = (V_1\ V_2)$ such that

1. AV_1 has no small singular values.
2. AV_2 is small.

In this case we say that A has approximate rank k, where k is the number of columns in V_1. In applications V_1 corresponds to the signal while V_2 corresponds to noise. We will call the space spanned by the columns of V_2 the ERROR SPACE.

As the signal changes, so does its error space. Since the *ab initio* computation of an error space is expensive, it is desirable to use the previously computed error space adaptively to approximate the new error space — a process that is generally called UPDATING. Our specific updating problem can be described as follows.

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. This work was supported in part by the Air Force Office of Scientific Research under Contract AFOSR-87-0188.

Given the error space of a matrix A compute the error space of the matrix

$$A_z = \begin{pmatrix} \beta A \\ z^{\mathrm{H}} \end{pmatrix},$$

where z is a new sample and $\beta \leq 1$ is a "forgetting factor" that damps out the effect of the previous samples. To simplify the exposition, we will take $\beta = 1$ in this paper (however, see the end of §2, where the problem of tolerances is treated).

The usual approach to computing error spaces has been via the singular value decomposition [4, 8]. Specifically, there are orthogonal matrices U and V such that

$$U^{\mathrm{H}} A V = \begin{pmatrix} \Sigma \\ 0 \end{pmatrix},$$

where

$$\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_p)$$

with

$$\sigma_1 \geq \cdots \geq \sigma_p.$$

The procedure for computing error spaces is to determine an integer k such that σ_k is above the noise level, while σ_{k+1} is below it. The columns of V corresponding to $\sigma_{k+1}, \ldots, \sigma_n$ then span the error space.

Although the singular value decomposition furnishes an elegant solution to the problem of calculating error spaces, it has two disadvantages: it is expensive to compute and it is difficult to update. The initial cost of computing a singular value decomposition would not be an objection, if the decomposition could be cheaply updated; however, all known updating schemes require on the order of p^3 operations (e.g., see [2]). Recently, abridged updating schemes that produce an approximate singular value decomposition have been proposed [7]. However, the effectiveness of this approach has not yet been demonstrated.

The difficulties in working with the singular value decomposition have sparked an interest in rank revealing QR decompositions, which decompose the matrix into the product of an orthogonal matrix, an upper triangular matrix, and a permutation matrix in such a way that the effective rank of the matrix is obvious [3]. However, a QR decomposition — even a rank-revealing one — does not provide an explicit basis for the error space. In this paper, we will consider an intermediary between the singular value decomposition and the QR decomposition — a two-sided orthogonal decomposition that we will call the URV decomposition — that has some of the virtues of both.

In the next section we will introduce the URV decomposition and its rank revealing variant. This section also contains a discussion of how to determine rank in the presence of errors. Since the updating will be accomplished by plane rotations, we give a brief review of their properties in §3. In the following section we will show how to compute a rank revealing URV decomposition of a triangular matrix. This special case will be used in §5 where we show how to update a rank revealing URV decomposition in such a way that it remains rank revealing. In §6 we will show that the updating algorithm can be implemented on a linear array of processors in such a way that it runs in $O(p)$ time. Finally, in the last sections we will make some general observations on the updating algorithm.

Throughout this paper $\|\cdot\|$ will denote the Euclidean vector norm and the Frobenius matrix norm defined by

$$\|A\|^2 = \sum_{i,j} |a_{ij}|^2.$$

The smallest singular value of a matrix A will be written $\inf(A)$.

2. URV Decompositions

Suppose for the moment that A has rank k. Then there are orthogonal matrices U and V such that

$$A = U \begin{pmatrix} R & 0 \\ 0 & 0 \end{pmatrix} V^{\mathrm{H}}, \tag{2.1}$$

where R is an upper triangular matrix of order k. We will call this decomposition a URV decomposition. Unlike the singular value decomposition the URV decomposition is not unique; in fact, the singular value decomposition is itself a URV decomposition. However, we will be concerned with the case where R is not diagonal but fully triangular.

Now suppose that A is nearly of rank k in the sense that its singular values satisfy

$$\sigma_1 \geq \cdots \geq \sigma_k > \sigma_{k+1} \geq \cdots \geq \sigma_p,$$

where σ_k is large compared to σ_{k+1}. It can be shown that there is a URV decomposition of A of the form

$$A = U \begin{pmatrix} R & F \\ 0 & G \end{pmatrix} V^{\mathrm{H}}, \tag{2.2}$$

where

1. R and G are upper triangular,
2. $\inf(R) \cong \sigma_k$,
3. $\sqrt{\|F\|^2 + \|G\|^2} \cong \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_p^2}$.

The singular value decomposition itself is a example of such a decomposition, but there are many others. We will call any such decomposition a RANK REVEALING URV decomposition. From such a decomposition we can extract the error subspace, just as we did with the singular value decomposition. However, as we shall see, rank revealing URV decompositions are easier to compute and update than the singular value decomposition.

In practice the small singular values of A will come from noise, and the user must furnish a tolerance to distinguish them from the singular values associated with the signal. Unfortunately, the relation between noise and the small singular values is not simple, and the mathematical form of the tolerance is a matter of some delicacy.

We will adopt a very simple model. Suppose that A has the form

$$A = \hat{A} + E,$$

where $\hat{A}$ has rank exactly k. We will assume that the errors are roughly of the same size — say ϵ — so that when the forgetting factor is taken into account, E has the form

$$E = \begin{pmatrix} \beta^{n-1} e_1^{\mathrm{H}} \\ \beta^{n-2} e_2^{\mathrm{H}} \\ \vdots \\ e_n^{\mathrm{H}} \end{pmatrix},$$

where the components of the e_i are approximately ϵ in size. Let the columns of V_2 form an orthonormal basis for the error space of $\hat{A}$. Then our tolerance should approximate the norm of

$$AV_2 = EV_2$$

(remember $\hat{A}V_2 = 0$). Now the ith row of EV_2 consists of $p-k$ elements of size roughly $\beta^{n-i}\epsilon$. Consequently,

$$\|EV_2\|^2 \cong (p-k)\epsilon^2 \sum_{i=1}^{n} \beta^{2(n-i)} \leq \frac{(p-k)\epsilon}{1-\beta^2}.$$

BEFORE

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{0} & \mathtt{0} & \mathtt{X} & \mathtt{E} \\ \mathtt{X} & \check{\mathtt{X}} & \mathtt{X} & \mathtt{X} & \mathtt{0} & \mathtt{E} & \mathtt{E} \end{array}$$

AFTER

$$\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{X} & \mathtt{0} & \mathtt{X} & \mathtt{E} \\ \mathtt{X} & \mathtt{0} & \mathtt{X} & \mathtt{X} & \mathtt{0} & \mathtt{X} & \mathtt{E} \end{array}$$

Figure 3.1: Application of a Plane Rotation

Consequently the tolerance — call it tol — should be chosen so that

$$\mathrm{tol} \geq \sqrt{\frac{p-k}{1-\beta^2}}\,\epsilon. \tag{2.3}$$

Note that it is better to choose tol a little too large than too small. In the latter case, the dimension of the error space will be underestimated. On the other hand, if the tolerance is a little too large and there is a good signal to noise ratio, the tolerance will insinuate itself between the signal and the noise, and the dimension of the error space will be correctly estimated.

3. Plane Rotations

The chief computational tool of this paper is the plane rotation, which will be used to introduce zeros selectively into matrices to be updated. Since treatments of plane rotations are widely available (e.g., see [4]), we will not go into the numerical details here. Instead we will sketch the few basic facts needed to understand the updating algorithm and introduce some conventions for describing reductions based on plane rotations.

Figure 3.1 shows two rows of a matrix before and after the application of a plane rotation. The `X`'s represent nonzero elements, the `0`'s represent zero elements, and the `E`'s represent small elements. The plane rotation has been chosen to introduce a zero into the position occupied by the checked `X` in column 2. When the rotation is applied the following rules hold.

1. A pair of `X`'s remains a pair of `X`'s (columns 1 and 3).
2. An `X` and an `O` are replaced by a pair of `X`'s (column 4).
3. A pair of `O`'s remains a pair of `O`'s (column 5).
4. An `X` and an `E` are replaced by a pair of `X`'s (column 6).
5. A pair of `E`'s remains a pair of `E`'s (column 7).

The fact that a pair of small elements remains small (column 7) follows from the fact that a plane rotation is orthogonal and cannot change the norm of any vector to which it is applied. This is one of the key observations of this paper, since the point of the updating algorithm is to keep small elements small.

It requires about $4p$ multiplications and $2p$ additions to apply a plane rotation to two rows of length p. The multiplication count can be reduced by using so-called fast rotations; however, in either case the work involved is $O(p)$.

Premultiplication by a plane rotation operates on the rows of the matrix. We will call such rotations LEFT ROTATIONS. Postmultiplication by RIGHT ROTATIONS operates on the columns. Analogous rules hold for the application of a right rotation to two columns of a matrix.

When rotations are used to update a URV decomposition, the right rotations must be multiplied into V. To get a complete update of the decomposition, we must also multiply the left rotations into U. However, in most signal processing applications U is not needed, and this step can be omitted.

Algorithms that use plane rotations are best described by pictures. To fix our conventions, we will show how the matrix

$$\begin{pmatrix} R \\ x^{\mathrm{H}} \end{pmatrix}$$

can be reduced to upper triangular form by left rotations. Here we assume the R is itself upper triangular.

The reduction is illustrated in Figure 3.2. The elements of R and x^{H} are represented generically by r's and x's. The first step in the reduction is to eliminate the first element of x^{H} by a rotation that acts on x^{H} and the first row of R. The element to be eliminated has a check over it and the two rows that are being combined are indicated by the arrows to the left of the array.

$$
\begin{array}{ccccc}
\rightarrow & r & r & r & r \\
 & 0 & r & r & r \\
 & 0 & 0 & r & r \\
 & 0 & 0 & 0 & r \\
\rightarrow & \check{x} & x & x & x
\end{array}
\Longrightarrow
\begin{array}{ccccc}
 & r & r & r & r \\
\rightarrow & 0 & r & r & r \\
 & 0 & 0 & r & r \\
 & 0 & 0 & 0 & r \\
\rightarrow & 0 & \check{x} & x & x
\end{array}
\Longrightarrow
\begin{array}{ccccc}
 & r & r & r & r \\
 & 0 & r & r & r \\
\rightarrow & 0 & 0 & r & r \\
 & 0 & 0 & 0 & r \\
\rightarrow & 0 & 0 & \check{x} & x
\end{array}
\Longrightarrow
$$

$$
\begin{array}{ccccc}
 & r & r & r & r \\
 & 0 & r & r & r \\
 & 0 & 0 & r & r \\
\rightarrow & 0 & 0 & 0 & r \\
\rightarrow & 0 & 0 & 0 & \check{x}
\end{array}
\Longrightarrow
\begin{array}{cccc}
r & r & r & r \\
0 & r & r & r \\
0 & 0 & r & r \\
0 & 0 & 0 & r \\
0 & 0 & 0 & 0
\end{array}
$$

Figure 3.2: Reduction to Triangular Form

According to this notation, the second step combines the second row of R with x^{H} to eliminate the second element of the latter. Note that r_{21}, which is zero, forms a pair of zeros with the first component of x^{H}, so that the zero we introduced in the first step is not destroyed in the second step. The third and fourth steps of the reduction are similar.

For column operations with right rotations we will use an analogous notation. The main difference is that the arrows will point down to the columns being combined.

4. Deflation and Refinement

In applications we can expect occasional changes in the rank of the matrix A. An increase in rank usually makes itself felt in an obvious way. On the other hand, a decrease in rank can hide itself in the matrix R of the URV decomposition. Thus any updating algorithm must be able to detect rank degeneracy in R and act accordingly. In this section we will show how to compute a rank revealing URV decomposition of a $k \times k$ upper triangular matrix R.

The first step is to determine if R is defective in rank — that is, if $\inf(R)$ is less than a prescribed tolerance (see the discussion at the end of Section 2). This

$$\begin{array}{cccccccccc}
\rightarrow & \check{w} & & & 0 & & & 0 & & 0 \\
\rightarrow & w & & \rightarrow & \check{w} & & & 0 & & 0 \\
 & w & \Longrightarrow & \rightarrow & w & \Longrightarrow & \rightarrow & \check{w} & \Longrightarrow & 0 \\
 & w & & & w & & \rightarrow & w & & 1
\end{array}$$

Figure 4.1: Reduction of W

problem has been extensively studied under the rubric of condition estimation (see [6] for a survey), and there exist reliable algorithms that, given a triangular matrix R, produce a vector w of norm one such that with

$$b = Rw$$

we have

$$\eta \equiv \|b\| \cong \inf(R). \tag{4.1}$$

The algorithms generally require a single solution of a triangular system whose matrix is R and therefore require only $O(k^2)$ work.

The next step is to determine a sequence $V_1^{\mathrm{H}}, V_2^{\mathrm{H}}, \ldots, V_{k-1}^{\mathrm{H}}$ of rotations that eliminate the first $k-1$ component of w, so that w is zero except for its last component, which is one. The reduction is illustrated in Figure 4.1. Let $Q^{\mathrm{H}} = V_{k-1}^{\mathrm{H}} V_{k-2}^{\mathrm{H}} \cdots V_1^{\mathrm{H}}$ denote the product of the rotations obtained from this step.

Next we determine an orthogonal matrix P such that $P^{\mathrm{H}}RQ$ is upper triangular. This may be done by applying V_1, V_2, ..., V_{k-1} from the right to R as shown in Figure 4.2. The result of applying a rotation V_i to two columns of R is to place a nonzero element below the diagonal of R. A left rotation then eliminates this element and restores triangularity. The matrix P^{H} is the product of the left rotations. The entire process requires only $O(k^2)$ time.

The appearance of the quantities e in the last array of Figure 4.2 is meant to indicate that r_{kk} must be small on completion of the process. In fact, the norm of the last column is the number η defined by (4.1). To see that this is true, note that from (4.1)

$$b' \equiv P^{\mathrm{H}} b = (P^{\mathrm{H}} RQ)(Q^{\mathrm{H}} w) \equiv R'w'.$$

Since the last component of w' is one and all its other components are zero, we see that the last column of R is b'. Since the norm of a vector is unchanged when

$$\begin{array}{cccc} \downarrow & \downarrow & & \\ r & r & r & r \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \end{array} \Longrightarrow \begin{array}{ccccc} \rightarrow & r & r & r & r \\ \rightarrow & \check{r} & r & r & r \\ & 0 & 0 & r & r \\ & 0 & 0 & 0 & r \end{array} \Longrightarrow \begin{array}{cccc} & \downarrow & \downarrow & \\ r & r & r & r \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \end{array} \Longrightarrow$$

$$\begin{array}{ccccc} & r & r & r & r \\ \rightarrow & 0 & r & r & r \\ \rightarrow & 0 & \check{r} & r & r \\ & 0 & 0 & 0 & r \end{array} \Longrightarrow \begin{array}{cccc} & & \downarrow & \downarrow \\ r & r & r & r \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \end{array} \Longrightarrow \begin{array}{ccccc} & r & r & r & r \\ & 0 & r & r & r \\ \rightarrow & 0 & 0 & r & r \\ \rightarrow & 0 & 0 & \check{r} & r \end{array} \Longrightarrow \begin{array}{cccc} r & r & r & e \\ 0 & r & r & e \\ 0 & 0 & r & e \\ 0 & 0 & 0 & e \end{array}$$

Figure 4.2: Triangularization of RQ

it is multiplied by a unitary matrix, it follows that

$$\|b'\| = \|b\| = \eta.$$

Since $\eta \cong \inf(R)$, we have produced a URV decomposition that reveals that R has a small singular value. We can continue the process with the leading principle matrix of R' of order $k-1$.

Although this procedure solves the problem of deflating a rank deficient triangular matrix, it is possible to refine the decomposition, bringing it nearer to diagonality. The procedure begins by reducing the first $k-1$ elements in the last column of R to zero. This is done by a sequence of right rotations. The process is illustrated in Figure 4.3. Again only $O(k^2)$ time is required. However, the right rotations must be accumulated in V.

The second step in the refinement is to reduce R to upper triangular form by a sequence of left rotations as illustrated in Figure 4.4.

This second step also requires $O(k^2)$ time, and the final matrix is clearly in URV form. However more can be said: the norm of the last column is less that the absolute value of the (k,k)-element before the refinement was begun. To see this, note that in the first step of the refinement procedure, the (k,k)-element does not increase. Thus at the end of the first step the norm of the last column is less than

$$
\begin{array}{cccc}
 & & \downarrow & \downarrow \\
r & r & r & e \\
0 & r & r & e \\
0 & 0 & r & \check{e} \\
0 & 0 & 0 & e
\end{array}
\Longrightarrow
\begin{array}{cccc}
 & \downarrow & & \downarrow \\
r & r & r & e \\
0 & r & r & \check{e} \\
0 & 0 & r & 0 \\
0 & 0 & e & e
\end{array}
\Longrightarrow
\begin{array}{cccc}
\downarrow & & & \downarrow \\
r & r & r & \check{e} \\
0 & r & r & 0 \\
0 & 0 & r & 0 \\
0 & e & e & e
\end{array}
\Longrightarrow
\begin{array}{cccc}
 & & & \\
r & r & r & 0 \\
0 & r & r & 0 \\
0 & 0 & r & 0 \\
e & e & e & e
\end{array}
$$

Figure 4.3: Reducing the Last Column

$$
\begin{array}{ccccc}
\rightarrow & r & r & r & 0 \\
 & 0 & r & r & 0 \\
 & 0 & 0 & r & 0 \\
\rightarrow & \check{e} & e & e & e
\end{array}
\Longrightarrow
\begin{array}{ccccc}
 & r & r & r & e \\
\rightarrow & 0 & r & r & 0 \\
 & 0 & 0 & r & 0 \\
\rightarrow & 0 & \check{e} & e & e
\end{array}
\Longrightarrow
\begin{array}{ccccc}
 & r & r & r & e \\
 & 0 & r & r & e \\
\rightarrow & 0 & 0 & r & 0 \\
\rightarrow & 0 & 0 & \check{e} & e
\end{array}
\Longrightarrow
\begin{array}{cccc}
r & r & r & e \\
0 & r & r & e \\
0 & 0 & r & e \\
0 & 0 & 0 & e
\end{array}
$$

Figure 4.4: Reducing the Last Row

the absolute value of the (k,k)-element before the refinement was begun. The same is true at the end of the second step, since this step uses only right rotations and cannot change the norm of a column.

In practice, the refinement step can reduce the size of the part last column lying above the diagonal so that it is insignificant compared to the diagonal element. The effect of this is to polish the approximation to the error space. Whether the refinement is worth its cost will have to be determined experimentally. Preliminary runs indicate that the refined version is preferable to the unrefined version when it is used with the MUSIC algorithm [5].

5. Updating a URV Factorization

In this section we will show how to update a rank revealing URV decomposition of A when a row z^{H} is appended. Specifically we will suppose that A has the URV decomposition (2.2), where V is known. To decide what is small we will suppose we have a user supplied tolerance, tol, and that

$$\nu \stackrel{\mathrm{def}}{=} \sqrt{\|F\|^2 + \|G\|^2} \leq \mathrm{tol}.$$

As is implied by (2.3), the tolerance may depend on dimensions of A and R and the size of the forgetting factor.

The first step is to compute

$$(x^{\mathrm{H}}\; y^{\mathrm{H}}) = z^{\mathrm{H}} V,$$

where x is of dimension k — i.e., the order of R. Our problem then becomes one of updating the matrix

$$\hat{A} = \begin{pmatrix} R & F \\ 0 & G \\ x^{\mathrm{H}} & y^{\mathrm{H}} \end{pmatrix}.$$

There are two cases to consider. The first, and simplest occurs when

$$\sqrt{\nu^2 + \|y\|^2} \leq \mathrm{tol}. \tag{5.1}$$

In this case we reduce $\hat{A}$ to triangular form by a sequence of left rotations as in Figure 3.2. Since the new value of ν will be given by the left-hand side of (5.1), we are assured that within our tolerance the rank cannot increase. However, it is

$$
\begin{array}{cccc}
 & & \downarrow & \downarrow \\
f & f & f & f \\
g & g & g & g \\
0 & g & g & g \\
0 & 0 & g & g \\
0 & 0 & 0 & g \\
y & y & y & \check{y}
\end{array}
\Longrightarrow
\begin{array}{c|cccc}
 & f & f & f & f \\
 & g & g & g & g \\
 & 0 & g & g & g \\
\rightarrow & 0 & 0 & g & g \\
\rightarrow & 0 & 0 & \check{g} & g \\
 & y & y & y & 0
\end{array}
\Longrightarrow
\begin{array}{cccc}
 & \downarrow & \downarrow & \\
f & f & f & f \\
g & g & g & g \\
0 & g & g & g \\
0 & 0 & g & g \\
0 & 0 & 0 & g \\
y & y & \check{y} & 0
\end{array}
\Longrightarrow
\begin{array}{c|cccc}
 & f & f & f & f \\
 & g & g & g & g \\
\rightarrow & 0 & g & g & g \\
\rightarrow & 0 & \check{g} & g & g \\
 & 0 & 0 & 0 & g \\
 & y & y & 0 & 0
\end{array}
\Longrightarrow
$$

$$
\begin{array}{cccc}
\downarrow & \downarrow & & \\
f & f & f & f \\
g & g & g & g \\
0 & g & g & g \\
0 & 0 & g & g \\
0 & 0 & 0 & g \\
y & \check{y} & 0 & 0
\end{array}
\Longrightarrow
\begin{array}{c|cccc}
 & f & f & f & f \\
\rightarrow & g & g & g & g \\
\rightarrow & \check{g} & g & g & g \\
 & 0 & 0 & g & g \\
 & 0 & 0 & 0 & g \\
 & y & 0 & 0 & 0
\end{array}
\Longrightarrow
\begin{array}{cccc}
f & f & f & f \\
g & g & g & g \\
0 & g & g & g \\
0 & 0 & g & g \\
0 & 0 & 0 & g \\
y & 0 & 0 & 0
\end{array}
$$

Figure 5.1: Reduction of y^{H}

possible for the rank to decrease. Hence we must check and possibly reduce R as described in §4. The time required for this case is $O(p^2)$.

If (5.1) is not satisfied, there is a possibility that there is an increase in rank. Since the increase in rank can be at most one, the problem is to transform the matrix to upper triangular form without destroying all the small values in F and G.

The first step is to reduce y^{H} so that it has only one nonzero component and G remains upper triangular. The reduction is illustrated in Figure 5.1. Since R and x^{H} are not involved in this part of the reduction, we show only F, G, and y^{H} (n.b., the f's in the figure represent entire columns of F).

Finally, the entire matrix

$$\begin{matrix} R & f & f & f & f \\ 0 & g & g & g & g \\ 0 & 0 & g & g & g \\ 0 & 0 & 0 & g & g \\ 0 & 0 & 0 & 0 & g \\ x^{\mathrm{H}} & y & 0 & 0 & 0 \end{matrix}$$

is reduced to triangular form in the usual way to give a matrix of the form

$$\begin{matrix} R & y & f & f & f \\ 0 & y & g & g & g \\ 0 & 0 & g & g & g \\ 0 & 0 & 0 & g & g \\ 0 & 0 & 0 & 0 & g \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \tag{5.2}$$

Then k is increased by one, and the new R is checked for degeneracy and if necessary reduced as described in §2. The result is the updated URV decomposition.

6. Parallelization

In this section we will show that the updating algorithm can be implemented on an array of p processors to yield an $O(p)$ algorithm. To simplify matters, we will consider shared memory implementation; however, it will be clear that the algorithms can be implemented on a linear array of distributed memory processors, provided fine-grain communication is sufficiently fast — e.g., on a linear systolic array.

Since we are concerned with the existence of a parallel algorithm, rather than in the details of a particular implementation, we will use the method of PRECEDENCE DIAGRAMS. The idea is to write down an order in which operations can be performed consistently and then assign operations to processors in such a way that no two simultaneous operations are performed by the same processor. In our case, the assignments will amount to making each processor responsible for a row of the matrix and its neighbors.

$$
\begin{array}{ccccccccc}
\circ & 1 & \circ & 2 & \circ & 3 & \circ & 4 & \circ \\
\circ & 2 & \circ & 3 & \circ & 4 & \circ & 5 & \circ \\
\circ & 3 & \circ & 4 & \circ & 5 & \circ & 6 & \circ \\
\circ & 4 & \circ & 5 & \circ & 6 & \circ & 7 & \circ \\
\circ & 5 & \circ & 6 & \circ & 7 & \circ & 8 & \circ
\end{array}
$$

Figure 6.1: Precedence Diagram for $VP_1 \cdots P_{p-1}$

We begin with the updating of V by rotations. Specifically we desire to compute $VP_1 \cdots P_{p-1}$, where P_i combines columns i and $i+1$ of V. Figure 6.1 shows a precedence diagram for this computation for $p = 5$. The circles represent elements of V. The numbers between any two circles represent the time at which a rotation can be applied to the two corresponding elements. We will call these numbers TICKS.

The increasing ticks in the first row reflect the fact that P_i must be applied before P_{i+1}. From an arithmetic point of view, the ticks need not increase as we go down a column, since a right rotation can be applied simultaneously to all the elements of a pair of columns. We have allowed the ticks to increase to reflect the realities of communication: on a shared memory system, there will be contention along a column as the processors attempt to access the same rotation; in a distributed memory system the rotation must be passed down the column. In general, the method of precedence diagrams does not require one to write down the best diagram — only a correct one.

Figure 6.2 shows the assignment of operations to processors. Operations between two horizontal lines are performed by the same processor, which, as we have observed, amounts to an assignment of the elements of V by rows to the processors. The diagram makes it obvious that the updating of V can be performed with p processors in about $2p$ ticks.

Let us now consider a more complicated case: the reduction of R in Figure 4.2. The construction of an appropriate precedence diagram is simplified by the fact that the right rotations can be applied in their entirety before the left rotations are generated and applied. Thus we can construct the the diagram for the right rotations and then fill in the left rotations in a suitable manner. The result is shown in Figure 6.3. A number between two circles in a row represents a right

$\circ$	1	$\circ$	2	$\circ$	3	$\circ$	4	$\circ$
$\circ$	2	$\circ$	3	$\circ$	4	$\circ$	5	$\circ$
$\circ$	3	$\circ$	4	$\circ$	5	$\circ$	6	$\circ$
$\circ$	4	$\circ$	5	$\circ$	6	$\circ$	7	$\circ$
$\circ$	5	$\circ$	6	$\circ$	7	$\circ$	8	$\circ$

Figure 6.2: Assignment for $VP_1 \cdots P_{p-1}$

$\circ$	1	$\circ$	2	$\circ$	3	$\circ$	4	$\circ$	5	$\circ$
6		7		8		9		10		11
$\circ$	2	$\circ$	3	$\circ$	4	$\circ$	5	$\circ$	6	$\circ$
		8		9		10		11		12
		$\circ$	4	$\circ$	5	$\circ$	6	$\circ$	7	$\circ$
				10		11		12		13
				$\circ$	6	$\circ$	7	$\circ$	8	$\circ$
						12		13		14
						$\circ$	9	$\circ$	10	$\circ$
								14		15
								$\circ$	11	$\circ$

Figure 6.3: Reduction of w and R

$\circ$	14	$\circ$	12	$\circ$	10	$\circ$	8	$\circ$	6	$\circ$
15		16		17		18		19		20
$\circ$	13	$\circ$	11	$\circ$	9	$\circ$	7	$\circ$	5	$\circ$
		12		13		14		15		16
		$\circ$	10	$\circ$	8	$\circ$	6	$\circ$	4	$\circ$
				9		10		11		12
				$\circ$	7	$\circ$	5	$\circ$	3	$\circ$
						6		7		8
						$\circ$	4	$\circ$	2	$\circ$
								3		4
								$\circ$	1	$\circ$

Figure 6.4: Reduction of y and G

rotation; between two circles in a column, a left rotation. From this diagram it is seen that the reduction can be carried out in about $3p$ ticks.

Finally, we consider the most difficult task of all: the construction of a precedence diagram for the reduction of y and G in Figure 5.1. The difficulty is that the right and left rotations must be interspersed; for if they are not, the right rotations will fill out the bottom half of G. The following precedence diagram is for the matrix G, the matrix F being handled independently by other processors. This reduction requires about $4(p-k)$ ticks plus k ticks to apply the rotations to F.

The other parts of the updating algorithm can be analyzed similarly. In all cases a row oriented assignment of elements to processors results in parallel implementations that take $O(p)$ ticks.

7. Comments

We have shown how to update a rank revealing URV factorization of a matrix using plane rotations. In this concluding section we will try to put our contribution in perspective.

Initialization. An attractive feature of the algorithm is that it requires no initial calculation of a decomposition. Instead one starts with a degenerate URV decomposition, in which $k = 0$, $F = 0$, and $V = I$, and applies the updating algorithm as the rows of A enter. This is an important economy when it comes to implementing the algorithm with special purpose hardware.

Efficiency. The total operation count for one update is a multiple of p^2. This should be contrasted with a cost of $O(p^3)$ for a complete update of a singular value decomposition. In addition, the updating algorithm is rich in left rotations, which need not be accumulated in some applications. In fact, if rank changes are rare, the algorithm will seldom use any right rotations.

Reliability. The singular value decomposition is generally regarded as the most reliable technique for computing null spaces. However, the algorithm presented here is almost as reliable. The crux of the matter is the reliability of the condition estimator that produces the approximate null vector in (4.1). Although counterexamples exist for most condition estimators, a wealth of experiments and experience has shown them to be very reliable in real-life applications [6].

Effect of Rounding Errors. The algorithms proposed here are completely stable. Standard-rounding error analysis shows that the updated matrix is orthogonally equivalent to an original matrix that is perturbed by quantities proportional to the rounding unit times the norm of the matrix [10]. Moreover, the matrix V deviates only very slowly from orthogonality—the more so since V changes only when a change in rank is suspected.

Parallelization. We have seen that the algorithm can be implemented on a linear array of p processors so that it runs in order p time. On the other hand, the singular value decomposition requires p^2 processors to achieve the same updating time.

Numerical Results. In [1], the algorithm has been used with the MUSIC algorithm to estimate directions of arrival from simulated data. The algorithm performs well, estimating the rank correctly and providing a sufficiently accurate error subspace to obtain the directions of arrival.

Availability. An experimental FORTRAN version of the algorithm is available by anonymous ftp at `thales.cs.umd.edu` in the file `pub/reports/uast.f`.

Acknowledgements

This work has its origins in a stimulating workshop on the singular value decomposition in signal processing, organized by R. J. Vaccaro. I am indebted to Frank

Luk for jogging my mind out of the QR rut by mentioning complete orthogonal decompositions.

References

[1] G. Adams, M. F. Griffin, and G. W. Stewart (1991). "Direction-of-Arrival Estimation Using the Rank-Revealing URV Decomposition." In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, Washington, DC. To appear.

[2] J. R. Bunch and C. P. Nielsen (1978). "Updating the Singular Value Decomposition." *Numerische Mathematik*, **31**, 111–129.

[3] T. F. Chan (1987). "Rank Revealing QR Factorizations." *Linear Algebra and Its Applications*, **88/89**, 67–82.

[4] G. H. Golub and C. F. Van Loan (1989). *Matrix Computations* (2nd ed.). Johns Hopkins University Press, Baltimore, Maryland.

[5] M. F. Griffin (1990). Personal Communication.

[6] N. J. Higham (1987). "A Survey of Condition Number Estimation for Triangular Matrices." *SIAM Review*, **29**, 575–596.

[7] Mark Moonen, Paul Van Dooren, and Joos Vandewalle (1990). "Combined Jacobi-type Algorithms in Signal Processing." In R. J. Vaccaro, editor, *Proceedings of the 2nd International Workshop on SVD and Signal Processing*, pages 83–88. Elsevier Science Publishers, Amsterdam. To appear. A preliminary version was circulated at the conference.

[8] G. W. Stewart (1974). *Introduction to Matrix Computations.* Academic Press, New York.

[9] R. J. Vaccaro, editor. *Proceedings of the 2nd International Workshop on SVD and Signal Processing*, Amsterdam. Elsevier Science Publishers. To appear. A preliminary version was circulated at the conference.

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