UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

The RRQR Factorization and Its Systolic Implementation

F. Lorenzelli
P. C. Hansen
T.F. Chan
K. Yao

December 1992

CAM Report 92-49

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

The RRQR Factorization and Its Systolic Implementation

F. Lorenzelli* P. C. Hansen† T. F. Chan‡ K. Yao§

December 1992

Abstract

The rank revealing QR factorization of a rectangular matrix is a potentially useful tool from linear algebra in many signal processing applications, since it essentially yields all the necessary information to solve rank deficient least-squares problems, to compute signal and noise subspaces, *etc.* In this paper we first review some of these applications, and then we discuss some issues of rank revealing QR factorization algorithms for implementation on systolic arrays. Next, we describe a particular systolic algorithm and we give a detailed discussion of its implementation on a systolic array. Finally, we consider the performance of the array and we compare it with the performance of systolic arrays for computing a related decomposition, namely, the singular value decomposition.

*Department of Electrical Engineering, University of California, Los Angeles, CA 90024. The work of this author was supported by a UC MICRO grant and the NSF grant NCR - 8814407. E-mail: lorenz@ee.ucla.edu

†UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby, Denmark. The work of this author was supported by a NATO Collaborative Research Grant 5-2-05/RG900098. E-mail: unipch@wuli.uni-c.dk

‡Department of Mathematics, University of California, Los Angeles, CA 90024. This author was supported by a NATO Collaborative Research Grant 5-2-05/RG900098. E-mail: chan@math.ucla.edu

§Department of Electrical Engineering, University of California, Los Angeles, CA 90024. This author was supported by a UC MICRO grant and the NASA/Ames grant NCC 2-374. E-mail: yao@ee.ucla.edu

1 Introduction

1.1 Motivation

In many fields of signal and image processing, control, and telecommunication there is much interest today in the numerical techniques offered by linear algebra. As problems grow in complexity and size, numerical issues such as stability, problem sensitivity, and error propagation become increasingly important.

The singular value decomposition (SVD) of a rectangular matrix is one of the techniques which have proven useful in many engineering applications, ranging from least squares problems, system identification, subset selection, direction of arrival estimation, linear regression, *etc.* [1, 26, 25, 35, 41, 22, 48, 32]. Very often in these applications, a reliable estimate of the rank of the matrix, as well as a good approximate basis for the numerical null space, are required. This is the case, *e.g.*, in rank-deficient least squares problems, estimation of the number of bearers in a radar signal, *etc.* [34, 54].

Unfortunately, the computation of the SVD is a costly numerical procedure. During the last few years alternative methods to the SVD—based on the QR factorization (QRF), which requires much less computational effort—have therefore emerged. In particular, much effort has been spent in refining the QRF algorithm such that it yields rank and null-space estimates which are almost as reliable and accurate as those from the SVD. Such QRFs are termed Rank-Revealing QR (RRQR) Factorizations. Some applications of QRFs in signal processing and linear algebra include [5, 6, 12, 21, 23, 24, 46].

Even on parallel processors, the complete computation of an SVD can be burdensome, in terms of computation count and execution time. Also QRFs may not lend themselves to a systolic implementation, in particular when they require column interchanges, as is the case when pivoting is adopted. The VLSI realization of QRFs without column reshuffling is, on the other hand, very simple. The main purpose of this paper is to present a version of the RRQR algorithm which is suited for implementation on a VLSI systolic array, in the sense that the column movement required by the original RRQR algorithm is simplified, as explained later, in §3.

However, before discussing the RRQR algorithm and its implementation, we shall first briefly introduce the SVD and least-squares problems. Then, as motivation for our analysis we shall also survey some applications in which numerical rank and null-space estimation are indeed needed as part of the computations.

1.1.1 The singular value decomposition

In this paper, we assume that the matrix A is $m \times n$ with $m \geq n$. Then the SVD of A is a decomposition in the following form

$$A = U \Sigma V^H,$$

where the superscript ^{H} denotes conjugate transpose. Here, U and V are matrices of dimensions $m \times n$ and $n \times n$, respectively, with orthonormal columns, and Σ is an $n \times n$ diagonal matrix with diagonal elements σ_i which are the singular values of A. They are non-negative and ordered in non-increasing fashion, *i.e.*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

The singular values are very useful for defining the numerical rank of A. Basically, the numerical rank r of a matrix is the number of columns which are guaranteed to be linearly independent. For example, in a direction-of-arrival problem, r represents the number of emitters. A strict definition of the numerical rank is easily given in terms of the singular values of A: then r is simply the number of "large" singular values of A. Typically, from the particular application one has some *a priori* threshold τ below which the singular values can be considered as numerically zero (representing noise only), [24, §5.5.8] [51, 47, 29]. In other words, the numerical rank r of A is defined by

$$\sigma_r \geq \tau > \sigma_{r+1}.$$

Then it is convenient to partition the matrices U, Σ and V as follows:

$$A = (U_S \; U_N) \begin{pmatrix} \Sigma_S & 0 \\ 0 & \Sigma_N \end{pmatrix} (V_S \; V_N)^H, \tag{1}$$

where U_S and V_S have r columns, and where Σ_S is $r \times r$. The subscripts S and N denote "signal" and "noise", respectively, for reasons that will become clear in §1.1.3.

We note in passing that the SVD can in principle be computed from an eigendecomposition of the symmetric cross-product matrix $A^H A$ because $A^H A = V \Sigma^H \Sigma V^H$. However, notice that the condition number of $A^H A$ is the square of that of A. Since A tends to have a large condition number in some of the applications that we have in mind, the cross-product algorithms cannot be recommended due to their potential numerical instability, and special SVD algorithms therefore must be used.

1.1.2 Least squares problems

Least squares problems appear in a vast number of applications, such as system identification, parameter estimation, subspace fitting, linear regression, and so on. Given an $m \times n$ matrix A with $m > n$, the problem is, formally, to find the n-vector x that solves

$$\min \|A\,x - b\|_2, \tag{2}$$

where the 2-norm $\|A\,x - b\|_2$ is the Euclidean length of the residual vector $A\,x - b$. This minimization problem can in principle be solved using the so-called normal equation approach, in which one solves the square system

$$A^H A\,x = A^T b,$$

but it is well known that in practice this computation can be very sensitive to rounding errors and perturbations in the data matrix columns [24, §5.3.9]. Therefore, the normal equations are not suited for ill-conditioned problems where some of the n columns of A are almost linearly dependent on the other columns [24, §5.3.9]. The standard QR factorization of A,

$$A = QR,$$

where Q is $m \times n$ and has orthonormal columns and R is $n \times n$ upper triangular, provides a numerically safer way to proceed, since in this case it is sufficient to solve the following triangular system of linear equations by backsubstitution,

$$R\,x = Q^H b.$$

However, when the matrix A is highly ill-conditioned or numerically rank deficient, then also this approach is very unstable and a different approach is needed.

Assume now that r is the numerical rank of A. Then one costly approach is to compute a truncated SVD solution, defined as

$$x_{TSVD} = V_S \, \Sigma_S^{-1} U_S^H b,$$

where V_S, Σ_S and U_S are defined in (1), [27]. However, a pivoted QR factorization can also be used. Suppose that it is possible to find a column permutation, represented by the matrix Π, such that in the QRF of $A\Pi$, *i.e.* $A\Pi = QR$, we have

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \begin{matrix} \updownarrow & r \\ \updownarrow & n-r \end{matrix}$$

where the $r \times r$ submatrix R_{11} is well-conditioned and the $(n-r) \times (n-r)$ submatrix R_{22} has "small" elements. A natural way to proceed is then to use only the first r rows of the triangular factor R in (2) and define the so-called *truncated QR solution* as

$$x_{TQR} = \Pi P \begin{pmatrix} \hat{R}_{11}^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q^H \, b,$$

where P is an orthogonal transformation such that $(R_{11}, R_{12})P = (\hat{R}_{11}, 0)$. An alternative approach often used in practice is to use only the $r \times r$ leading submatrix R_{11} instead and define the *basic solution* as

$$x_B = \Pi \begin{pmatrix} R_{11}^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q^H \, b.$$

The two solutions x_{TQR} and x_B have almost the same residual, in that we have

$$\|(A\, x_B - b) - (A\, x_{TQR} - b)\|_2 \leq \|R_{22}\|_2 \, \|R_{11}^{-1}\|_2 \, \|b\|_2.$$

They mainly differ in that x_{TQR} has all nonzero elements while x_B has zero elements corresponding to those elements of A that are considered linearly dependent of the others.

As we see, in rank-deficient or ill-conditioned least squares problems the QR decomposition must display the numerical rank r of A in order to produce well-defined solutions x_{TQR} and x_B. It is therefore desirable to resort to a rank revealing version of the QRF, such as the one considered in this paper.

1.1.3 Direction-of-arrival estimation

The direction-of-arrival (DOA) estimation problem is encountered in radar/sonar applications. In this environment it is assumed that a number of antennae in the far field receive a superposition of narrow-band signals, of same known center frequency, emitted by sources located at different angular directions. The underlying assumptions are that both source signals and noise are stationary zero-mean random processes, the noise being temporally and spatially white. The physical location of the antennae is known. The goal is to determine the number of emitters and their angular position. For this purpose some of the most popular parameter estimation techniques have been used, such as the so-called subspace fitting methods (*e.g.* MUSIC [50, 51], WSF [42], ESPRIT [49], SR [53], *etc.*). In all of these techniques, a reasonable estimate of either the "signal" or the "noise" subspace is required. As an example, MUSIC estimates the directions of arrival by determining the minima of a function of the form

$$f(\vartheta) = a^H(\vartheta) V_N V_N^H a(\vartheta) = \|V_N^H a(\vartheta)\|_2^2. \tag{3}$$

Here, $a(\vartheta)$ is the "steering vector" as function of the angle ϑ, and the columns of the matrix V_N span the estimate of the "noise" subspace which, in linear algebra terms, is identical to the numerical null-space of the covariance matrix at hand. For stability and computational reasons, the numerically best choice for V_N is a matrix whose columns form an orthonormal basis for the noise subspace, even though this is not a necessary requirement.

If the emitters are located at distinct angular positions and the signal-to-noise ratio is sufficiently high, then the most reliable way to obtain the required matrix V_N is to resort to the use of the SVD method. Suppose the columns of the matrix A represent the outputs of the n antennae. Each column is a vector of m elements, with $m > n$, corresponding to the m observations in time. The matrix A ends up showing a numerical rank equal to the number of emitters (say r), and its SVD is given by (1). In absence of noise, the $n - r$ smallest singular values are zero (*i.e.*, $\Sigma_N = 0$), whereas if the noise is white Gaussian with variance σ^2, all the singular values are shifted by σ and $\Sigma_N = \sigma I_{n-r}$. It can be easily shown that the covariance matrix $C = E\{A^H A\}/m$ can be written

as

$$C = C_S + C_N = V_S \Sigma_S^2 V_S^H + V_N \Sigma_N^2 V_N^H,$$

where C_S is a rank-r positive semi-definite matrix, with r "large" eigenvalues (*i.e.*, $\gg \sigma^2$). It is clear that V_N spans the numerical null-space of C and is therefore what we need in eq. (3).

As we have seen, the rank determination is a fundamental requirement in order to determine the dimensionality of the signal and noise subspaces. Moreover, the null-space calculation is essential. The SVD is an expensive procedure and alternative, cheaper techniques (such as the algorithm of §2) are therefore desirable. Methods to reduce the complexity of the MUSIC estimate and its update, based on the Chan/Foster algorithm of §2 have been proposed in [43] and [5].

1.1.4 Subset selection and collinearity

Consider the case in which we have n sensors. Their outputs, in numerical form, are stored as columns in a matrix A. It may occur that the data provided by a few sensors are actually superfluous or bring very little additional information. If this is the case, the matrix A will contain a number of columns which are very "similar" to each other (if not completely equal, in case of no noise). In linear algebraic terms, it is said that these columns of A are "collinear." In practice, it may be useful to determine which of the sensors are redundant and then procede to discard them. The presence of disturbance in the observations make the problem more difficult. A possible approach is to find a suitable permutation Π of the columns of A so as to concentrate the "energy" in the first (say) r columns of $A\Pi$. In other words, if r is the number of non-redundant sensors, the aim is to find a column permutation Π such that the first r columns of $A\Pi$ are as well-conditioned as possible. The basic solution x_B of least squares problems (§1.1.2) is related to the subset selection problem in the sense that the elements of x_B corresponding to the linearly dependent columns of A are forced to zero.

A QRF with ordinary column pivoting tends to find the "most linearly independent" columns of A, but is not guaranteed to succeed in doing so. SVD-based algorithms have been proposed by Golub, Klema and Stewart [22, 24] and Van Huffel and Vandewalle [31]. The problem with these techniques is that they require both an SVD and a QRF and are therefore very computationally

expensive. In particular, given the matrix A, the first step of the algorithm in [22] consists in computing the SVD of A, as

$$A = U\Sigma V^H.$$

The numerical rank r of A is revealed if Σ, the diagonal matrix that holds A's singular values, can be partitioned as in (1). The second step constructs the permutation matrix Π such that the bottom right $(n-r)\times(n-r)$ submatrix of $\Pi^T V$ is well conditioned. As a result, the first r columns of $A\Pi$ are guaranteed to form a linearly independent set of columns of A.

The QRF with rank-revealing properties such as the one considered in this paper produces both r, the number of non-redundant sensors, and the desired permutation matrix Π and is significantly less burdensome than an SVD. Such a permutation is constructed in much the same way as in the SVD-based algorithm, but on the basis of the matrix of estimated null-vectors rather than the matrix of right singular vectors V. It is proved in [13] that even if this method and the SVD-based technique provide different permutations Π, the subspaces spanned by the first r columns of $A\Pi$ are in the two cases very close, under the hypothesis that there is a well defined gap between the r-th and $(r+1)$-st singular values of A.

1.1.5 Curve fitting by total least squares

A technique which is acquiring a considerable popularity in curve fitting, error-in-variables regression, spectral estimation, *etc.* is the so-called total least squares (TLS) method [23, 33, 44]. The kind of problems solved by this technique are of the same nature as the least squares problems of (2)), but both A and b are here considered affected by perturbations, and not just the vector b. In other words, given an overdetermined and incompatible system

$$Ax \approx b,$$

where both A and b are contaminated with errors, the goal is to introduce both a residual matrix ΔA and a residual vector Δb such that $\hat{A} = A+\Delta A$ and $\hat{b} = b+\Delta b \in \text{span}(\hat{A})$ and such that the TLS solution $\hat{x}$ solves the problem $\hat{A}\,\hat{x} = \hat{b}$. Moreover, to obtain a unique solution, the "approximation effort," *i.e.*, the Frobenius norm of the compound matrix $(\Delta A,\ \Delta b)$, is minimized. Hence, the

computational problem becomes:

$$\text{solve} \quad \hat{A}\,\hat{x} = \hat{b} \qquad \text{subject to} \qquad \min \|(\Delta A, \Delta b)\|_F.$$

The solution of a TLS problem typically requires a full or a partial SVD of the compound matrix (A, b). Assume that the matrix V in the SVD of (A, b) is partitioned such that

$$V = \begin{array}{c} \begin{array}{cc} \overset{r}{\longleftrightarrow} & \overset{l}{\longleftrightarrow} \end{array} \\ \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} \begin{array}{c} \updownarrow \\ \updownarrow \end{array} \begin{array}{c} n \\ 1 \end{array} \end{array},$$

where $l = n + 1 - r$, V_{11} is $n \times r$, and r is the numerical rank of A. Then the SVD-based solution to the TLS problem is given by [24, 17]

$$\hat{x} = -V_{12} V_{22}^H / \|V_{22}\|^2. \tag{4}$$

A computationally low cost and yet reliable solution can also be provided by the rank-revealing QRF of this paper. A solution of the same form as (4) can in fact be computed from the estimate of the null-space provided by the algorithm of §2. The two matrices V_{12} and V_{22} are now obtained after orthonormalization of the estimated null-space matrix, by means of the modified Gram-Schmidt process [13]. Again, the quantities used are the numerical rank r and the estimated null-space. In [13] it has been demonstrated that the accuracy of the result computed by the rank-revealing QRF is comparable with the accuracy of solutions produced by the SVD.

1.2 The rank-revealing QR factorization

As we have seen, the rank determination is very often a crucial problem in both numerical linear algebra and in many engineering applications. As we mentioned earlier the most reliable method for solving this problem is the SVD.

The computational effort involved in computing the SVD is quite large, however, and alternative methods that require less computational effort are therefore desirable. Most of these methods rely on a QRF with column pivoting [9, 35, 38]. Unfortunately, the ordinary QRF algorithm with column pivoting is not guaranteed to produce a reliable estimate of the numerical rank [24, 18, 19].

What one needs is a *rank revealing QR factorization* (RRQRF). Assume that there exists a column permutation Π such that the $m \times n$ matrix A with numerical rank r can be factorized as

$$A\Pi = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{5}$$

where Q is an $m \times n$ matrix with orthonormal columns, R is upper triangular, and Π is the permutation matrix produced by the specific factorization algorithm we are using. Then this particular QRF exhibits the numerical rank of the matrix A if it is possible to partition the upper triangular matrix R in (5) such that

- the submatrix R_{11} is $r \times r$
- the smallest singular value of R_{11} is of the order σ_r
- the largest singular value of the submatrix R_{22} is of the order σ_{r+1}

where σ_r and σ_{r+1} are singular values of A, for then we are guaranteed that R_{11} is well-conditioned, that the elements of R_{22} are small, and therefore that the rank r is revealed in the size of the submatrix R_{11}.

The crux of the computational problem is to identify a suitable permutation Π which is guaranteed to produce the desired "rank revealing" partitioning of the triangular factor R. Even though in many cases the ordinary QRF with column pivoting actually produces such a factorization, it may occasionally fail to do so.

L. V. Foster [18] and T. F. Chan [10] developed independently an algorithm which allows to choose a permutation Π that *guarantees* the QRF in (5) to reveal the rank of A. When implemented correctly, the overhead of this RRQRF algorithm, as compared to the ordinary QRF algorithm, is only $O((n-r)n^2)$ operations which is small whenever the rank deficiency $n-r$ is small compared to n. Moreover, as "by-products" of the algorithm which hardly require any extra overhead, the Chan/Foster algorithm produces an approximate basis for the null space of A, and it produces tight lower and upper bounds for the smallest $n-r+1$ smallest singular values.

The RRQRF can also be used in those cases in which the data matrix dynamically changes, by means of successive updates. The updating problem arises every time a new row is appended to

the data matrix A. This corresponds, for example, to the case in which A represents the outputs of m sensors in time. A form of update is obtained when the new row is appended to A, but the dimensionality of the matrix is maintained by erasing its first (oldest) row, as follows:

$$A = [a(1), a(2), \ldots, a(n)]^T \quad \rightarrow \quad A' = [a(2), a(3), \ldots, a(n), a(n+1)]^T.$$

It is well known that the problem of updating the SVD of a matrix is a particularly burdensome task [8]. The reason for this is that in order to update even one singular singular vector one needs all the singular values and vectors of the old A. Besides, each update requires $O(n^3)$ flops. An updating technique based on the RRQRF has been recently proposed by Bischof and Shroff [5], which requires only $O(n^2)$ work. The algorithm presented in [5] allows one to keep track of the rank of A, to update the triangular factor R, preserving its rank-revealing structure, and to compute the estimated null-space matrix, on the basis of the updated R. The possibility to tackle the updating problem in such an inexpensive way is one of the main advantages of the RRQRF algorithm versus that of the SVD. However, this issue will not be further considered in the rest of this paper.

1.3 Systolic array implementations

The purpose of this paper is to present a systolic array implementation of the RRQRF algorithm. In the literature, a number of different systolic realizations for the QRF without column pivoting have been described by various authors [40, 20], but to our knowledge nothing has been proposed for Chan's and Foster's version of the rank revealing QRF.

It is well known that systolic arrays represent a class of very efficient special purpose array processors for real-time applications which is also particularly amenable to VLSI architectures. The tremendous evolution of VLSI technology and design, along with the demand for high-speed processing, allows to exploit parallel processing concepts, more than ever before. Systolic and wavefront arrays have all the features that suit today's VLSI technology. In particular, among the main advantages of the systolic approach are simplicity, regularity and modularity of the layouts, local interconnection patterns and data pipelining, which in turn brings about the reduction of input/output bandwidth requirements.

There are obviously also drawbacks which have to be taken into account, such as the problem of clock skew in large architectures, the fixed size of the array processor and the constraints in the topology of the array. The first two problems can be solved by using careful design of the clock distribution network and by resorting to algorithm partitioning techniques—both heuristic and analytical. The third problem, namely the topology constraints, may impose limitations on the algorithms which can actually be mapped onto systolic structures. It is well known that, strictly speaking, only the so-called recursive iterative algorithms [45] can be made to correspond to systolic architectures. In the literature, though, some definitions of "systolic" have been relaxed, in order to include a wider class of algorithms. For instance, even though no data broadcast and no global communications are permitted *within* the array, long "wrap-around" connections are sometimes tolerated, in order to allow multi-stage or iterative processing.

With reference to the above discussion, two operations of the Chan/Foster algorithm (to be discussed in details in §2) do not appear to be well suited for systolic implementation: LINPACK condition estimation and cyclic shifts of the columns of R. The systolic array that we propose in this paper overcomes these difficulties by 1) using a simple "probabilistic" condition estimator based on the power method, which is sufficient for our purpose, and 2) by not computing a *complete* RRQRF, but only the submatrix R_{11} plus the necessary null vectors. An alternative approach, which makes use of processing elements of a different form and which is capable of computing both R_{11} and R_{12}, is considered in the Appendix.

We mention in passing that G. W. Stewart has proposed another strategy for computing a rank revealing factorization [52]. In addition to the condition estimation, this algorithm requires a sequence of both column and row rotations throughout the whole triangular matrix. The rotation parameters have to be computed outside of the main array, by an auxiliary processor. Besides, since the algorithm requires interleaved left and right rotations, Stewart's algorithm needs more operations and a longer execution time than the realization of the algorithm considered in the present paper. Also, the estimation of the null-space of the matrix, when needed, requires additional external computation, whereas in our algorithm the null-space comes almost free as a side-product of the procedure.

The implementation of the RRQRF that we propose in this paper requires $n(n+1)/2$ processors and $O(n)$ external buffers, for a problem of order n. An external processor is also required for a few intermediate operations. Given a preliminary QRF of A, the execution time for the algorithm is $O(n(n-r))$, where r is A's numerical rank. This should be compared with the existing systolic arrays for SVD which require $O(n^2)$ processors, and for which both the processors and their interconnection network is more complex than for our RRQRF algorithm. The execution time for the SVD systolic arrays is $O(N_S n)$, where N_S is of the order n. There are also Jacobi-type systolic SVD arrays for which N_S is of the order $\log n$, but in order to compute an accurate null-space these methods require either additional computations to refine the null-space or an additional n^2 array to accumulate the orthogonal transformations.

Our paper is organized as follows. The Chan/Foster RRQRF algorithm is outlined in Section 2. Next, in Section 3 a systolic implementation is described in terms of a multi-stage procedure. The array processor has the shape of a triangular array as in the classical array for QRF by Gentleman and Kung [20]. In Section 4 we mention several important implementation considerations. Finally, we illustrate the performance of the systolic RRQRF algorithm in Section 5. In the Appendix, an alternative systolic structure is briefly discussed.

2 The Chan/Foster RRQRF algorithm

As explained in the Introduction, the main goal of the RRQRF algorithm is to determine a column permutation Π which guarantees R_{22} in Eq. (5) to have "small" elements. To start with a simple case, assume that the numerical rank of the matrix A is $r = n - 1$. Let v_n be the right singular vector corresponding to A's smallest singular value σ_n, and let Π_n be the permutation that brings the largest element in absolute value of v_n to the bottom, as follows:

$$\Pi_n^T v_n \equiv w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \begin{matrix} \updownarrow & n-1 \\ \updownarrow & 1 \end{matrix},$$

where $|w_2| = \max_i |w_i|$. Then it can be shown [10] that the QRF of $A\Pi_n$ yields an $R_{22} = r_{nn}$ that satisfies

$$|r_{nn}| \leq \frac{\sigma_n}{|w_2|} \leq \sqrt{n}\,\sigma_n.$$

Here, the last inequality follows from the fact that $\|v_n\|_2 = 1 \Rightarrow |w_2| \geq 1/\sqrt{n}$.

Observe that we obtain not only a small $\|R_{22}\|_2$, which is our main goal, but also an estimate of the null-space of A, in the sense that $\|Av_n\|_2 \approx \sigma_n$ is small. This is a very useful side-product which renders the RRQRF algorithm extremely helpful in many real-life applications.

The RRQRF algorithm for matrices with arbitrary numerical rank is made up of a series of steps of the above type in which the smallest singular values of R are "peeled off" one at a time [10]. We present this algorithm below, including the threshold pivoting modification suggested in [4]. Here, and throughout the paper, $R_{11}^{(k)}$ denotes the leading $k \times k$ submatrix of the current R.

CHAN/FOSTER RRQRF ALGORITHM

1. Compute any QR factorization $A\Pi = QR$, for some initial Π and Q. The matrices Π, Q, and R are updated in the following steps.
2. "Peel off" the small singular values one at a time, starting with $k = n$:
   (a) Estimate the smallest singular value δ_k and the corresponding singular vector v_k of $R_{11}^{(k)}$.
   (b) Determine a large component of v_k, say $v_k(p)$, for which $|v_k(p)| \geq \rho \|v_k\|_\infty$, for a given $\rho \leq 1$.
   (c) Find the permutation Π_k that brings $v_k(p)$ to the last position of v_k by means of a cyclic shift of the elements $p, \ldots, k$.
   (d) Apply this permutation Π_k to the columns of the matrix $R_{11}^{(k)}$; this operation introduces a lower bidiagonal in R from column p to column k.
   (e) Restore R's upper triangular form by means of a series of Givens transformations.
   (f) Update Q, R, and Π, and let $k \leftarrow k - 1$.
3. Repeat steps 2a–2f until the singular value estimate in step 2a is above a prescribed threshold for the small singular values.

We emphasize that this algorithm is most efficient when the numerical rank r is close to the order of the matrix; *i.e.*, when $r \approx n$. There also exist versions of the RRQRF algorithm for low-rank

problems, such as L-RRQR [12]. Since the spirit of the low-rank algorithms is similar to the above algorithm, we will not address these algorithms in the sequel.

In addition to providing the desired RRQRF in Eq. (5), the above algorithm produces the following useful quantities, *cf.* [10, 11, 13].

1. Tight lower and upper bounds for the smallest $n - r + 1$ singular values, which is useful for checking the reliability of the computed numerical rank. These bounds are

$$\delta_k \leq \sigma_k \leq \|R_{22}^{(k)}\|_2, \qquad k = r, \ldots, n, \tag{6}$$

   where $R_{22}^{(k)}$ denotes the bottom $(n-k+1) \times (n-k+1)$ submatrix of R. The tightness of these bounds is proved in [10, Corollary 4.1].

2. A matrix W whose columns span an approximation to the null space of A. The matrix W is constructed during the algorithm by "stacking" the vectors $\Pi_k^T v_k$, properly augmented with zeros, into an upper trapezoidal matrix, and then multiplying with the final Π,

$$W = \Pi \begin{pmatrix} \Pi_{r+1}^T v_{r+1} & \cdots & \Pi_n^T v_n \\ 0 & & \end{pmatrix}. \tag{7}$$

   The quality of W as a basis for the null space of A is demonstrated in [11, Theorem 4.1].

The tight bounds in Eq. (6) are important primarily because they ensure that the algorithm computes the correct numerical rank of A. However, it is the null-space matrix W in (7) which has most interest in connection with many algorithms in signal processing that rely on a signal/noise-subspace approach, such as the techniques for parameter estimation mentioned in the Introduction. Recall, for instance, how the MUSIC estimate of the direction of arrival is intimately connected with a good estimate of the so-called noise subspace, *i.e.*, the numerical null-space of A.

In the RRQRF algorithm two operations are crucial, namely the singular value/vector estimation in step 2(a) and the column shift in step 2(d). The former operation can be obtained using the LINPACK condition estimator [16, 14] or Bischof's incremental condition estimation [2] or other sophisticated methods [30]. The latter operation, *i.e.*, the cyclic shift, moves the "most linearly dependent column" in $R_{11}^{(k)}$ to the back of the matrix. Unfortunately, neither of these operations are suited for a systolic array implementation: the sophisticated singular value/vector estimation

techniques require non-homogeneous computations in the construction of the singular vector estimate, and the cyclic shifts require global communication in the array. Therefore, other approaches have to be sought. In the next section we present our approach to these problems.

3 A systolic RRQRF algorithm

Our systolic algorithm for computing a RRQRF is essentially identical to the Chan/Foster algorithm, in that it consists of an initial QRF step and an iterative part. However, in the iterative part the estimation of the smallest singular value and the corresponding singular vector, as well as the pivoting and the restoration of the triangular form, differ somewhat from the details of the Chan/Foster algorithm.

3.1 Initial QRF step

The initial step, which in our algorithm is a QRF without pivoting, can be implemented on Gentleman and Kung's classical triangular array [20] (sometimes denoted "triarray" in the literature), as shown in Fig. 1, which implements Givens' method [24, §5] for computing a QRF. The data (skewed) enters the array row by row from the top. The diagonal elements compute the necessary Givens rotation parameters and, in the end, store the elements r_{ii} of the matrix R. The other processing elements (PE's) apply the rotation and transmit the rotation parameters. In the end, the PE on the i-th row and j-th column will store element r_{ij} of R. The operations of the processing elements are summarized in Table 1 in §4.2.

3.2 Estimation of the smallest singular value

During this step, the smallest singular value and the corresponding singular vector of $R_{11}^{(k)}$ are estimated. Here, $R_{11}^{(k)}$ denotes the leading $k \times k$ submatrix of R. The LINPACK condition estimator [16, 14] and the convex optimization approach [30] are not suited for systolic implementation because they require "external control" during the construction of the first singular vector estimate by means of a highly specialized and non-homogeneous "back-solve". Bischof's incremental condition estimation [2] is not suited either because of global communication or data broadcasting. Condition

estimators based on the theory of comparison matrices (*cf.* Algorithm 2.1 in [30]) do not require global communication, but they cannot be used because they do not provide the corresponding singular vector.

A simple and communication-wise "local" approach (except possibly for the need of feedback links) is to apply the simplest form of the power method *implicitly* to $[(R_{11}^{(k)})^H R_{11}^{(k)}]^{-1}$, *i.e.*:

POWER ALGORITHM

select $v^{(0)}$

for $h = 1, 2, \ldots$

$w^{(h)} \leftarrow (R_{11}^{(k)})^{-H} v^{(h-1)}$

$v^{(h)} \leftarrow (R_{11}^{(k)})^{-1} w^{(h)}$

$v^{(h)} \leftarrow v^{(h)} / \|v^{(h)}\|_2$

$\delta^{(h)} \leftarrow \|v^{(h)}\|_2^{-1/2}$

end

Here, $v^{(0)}$ is an appropriate starting vector. The vector $v^{(h)}$ needs to be rescaled in each iteration, since its norm grows with the square of the inverse of the smallest singular value in each step of the loop. Besides, the quantity $\delta^{(h)} = \|v^{(h)}\|_2^{-1/2}$ in each step is an estimate of the smallest singular value of $R_{11}^{(k)}$.

The described method belongs to the class of "probabilistic" condition estimation methods [15, 30] in that it starts from a unit norm vector $v^{(0)}$ whose structure has nothing to do with the matrix. It is, in a sense, chosen "randomly". For this type of estimates, it has been shown in [30] that the sequence $\{\delta^{(h)}\}$ is monotonically decreasing, approaching the true value of σ_n. The tightness of the estimate increases with h in an exponential fashion. The size n of the matrix also has an effect (although not very remarkable) on the goodness of the estimate, in the sense that the smaller n, the better the estimate. For details, *cf.* [30]. Numerical tests also show that $\delta^{(1)}$ is in most cases not greater than $3\sigma_n$ and a significant improvement in the estimate can be obtained already for $h = 2$.

For the purposes of our Algorithm, note that we only need an order-of-magnitude estimate of

the true smallest singular value and also for the singular vector, the estimate need not be extremely accurate (depending on the choice of the parameter ρ in step 2b of the RRQRF Algorithm). A reasonable policy is therefore to always take two iteration steps in the Power Algorithm, and avoid complicated stopping criteria. This method, in spite of its considerable simplicity, gives a good order-of-magnitude estimate, irrespective of the type of matrix and its size. It is actually shown in [30] that the quality of its estimates is very close to the one provided by the LINPACK condition estimator. The only case in which the performance of this method deteriorates is when some singular values are approximately equal. Nevertheless, the numerical experiments produced in [30] show that even in the case where all but one singular values are equal, the estimator does not break down catastrophically. Moreover, in case of several small singular values that are approximately equal, all that matters is that we compute some vector in the space spanned by the corresponding right singular vectors. These considerations justify our choice.

3.3 Pivoting and restoration of triangular form

In the Chan/Foster algorithm, the information obtained from the above estimation-step is now used to perform a cyclic shift that brings column p of the current R to the k-th position. The shift is followed by the annihilation of the subdiagonal introduced in R. In the systolic array, shifts and rotations could be performed at the same time, but this would require a more complex structure of the PE's or an increased processing time. In order to avoid both drawbacks and still be able to use the same triangular array employed in the previous steps, a simplified procedure is suggested here.

Our implementation allows us to immediately compute the submatrix R_{11}, along with an estimation of the numerical rank of A, its smallest singular values and the matrix W. The submatrix R_{12}, however, must be computed by a separate processor. The idea is the following: instead of shifting column p to the right end of the matrix, we simply eliminate it, by either overwriting it or by driving it out of the array. Observe that in the latter case it is actually possible to keep track of the submatrix R_{12} *externally*: the eliminated columns can be stacked and later updated by applying the same Givens rotations as performed in the triarray R_{11}. These additional operations can be performed on a sequential processor or another very simple systolic array whose processing

elements look like the off-diagonal cells of the Gentleman-Kung array.

The column shift in our systolic RRQRF algorithm proceeds this way: all the columns from 1 to $(p-1)$ are pushed to the right, leaving zeros behind on the diagonal; at the same time, the contents of the p-th column is either shifted to the right edge and out for external processing, or just overwritten and forgotten. At first sight, it appears that our choice of p should favor a small p in order to minimize the amount of shifts; however, since the shifts can be pipelined with the next operations, namely the restoration to triangular form, which always takes n time steps, the actual choice of p doesn't matter.

Once the column shift has been completed, the matrix stored in the array needs to be retriangularized. In principle, one could think to feed its entries to a smaller triangular array (note that the first column of such a matrix is all zero) and start the procedure all over again. It is obviously more convenient to use the same array, by simply "pushing down" the contents of the PE's. The same result can be obtained by inputting a new row from the top. A good choice for this additional row is a one followed by all zeros. To understand the rationale of this operation, consider the QR factorization of a matrix M of the type

$$M = \begin{pmatrix} 1 & 0^T \\ 0 & \tilde{M} \end{pmatrix}.$$

We have that

$$Q^T M = \begin{pmatrix} 1 & 0^T \\ 0 & \tilde{Q}^T \end{pmatrix} \begin{pmatrix} 1 & 0^T \\ 0 & \tilde{M} \end{pmatrix}$$

is triangular and so is the matrix $\tilde{Q}^T \tilde{M}$, which is what we wanted.

Thus, after the actual value of p is determined, the operations we suggest to perform are the following: (i) put the triarray in shift mode and shift columns 1 through $(p-1)$ to the right leaving zeros behind; (ii) put the triarray back in QR mode and input from the top, in the usual fashion, a row of all zeros except for a one at the very left. The array will then automatically zero out all the elements that a cyclic shift would have generated in the subdiagonal of R. In this way, we have "simulated" the cyclic shift, without actually performing it, at the expense of not having the p-th column stored in the final array.

In order to help visualize the algorithm, consider the following schemes concerning a 4×4 case in which we want to shift the second column to the back of the array (*i.e.*, $p = 2$ and $k = 4$). The

elements shown as x's are nonzero numbers, while the rest of the matrix consists of zeros. The first figure shows what a cyclic shift and subsequent Givens rotations will produce in the Chan/Foster algorithm (the x's inscribed in boxes are the sub-diagonal elements to annihilate, whereas the elements denoted by $\tilde{x}$ result from the Givens rotations):

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ & x_{22} & x_{23} & x_{24} \\ & & x_{33} & x_{34} \\ & & & x_{44} \end{pmatrix} \overset{\text{shift}}{\rightarrow} \begin{pmatrix} x_{11} & x_{13} & x_{14} & x_{12} \\ & x_{23} & x_{24} & x_{22} \\ & \boxed{x_{33}} & x_{34} & \\ & & \boxed{x_{44}} & \end{pmatrix} \overset{\text{Givens}}{\rightarrow} \begin{pmatrix} x_{11} & x_{13} & x_{14} & x_{12} \\ & \tilde{x}_{22} & \tilde{x}_{23} & \tilde{x}_{24} \\ & & \tilde{x}_{33} & \tilde{x}_{34} \\ & & & \tilde{x}_{44} \end{pmatrix}.$$

The second picture shows how we propose to proceed with our systolic array:

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ & x_{22} & x_{23} & x_{24} \\ & & x_{33} & x_{34} \\ & & & x_{44} \end{pmatrix} \overset{\text{shift}}{\rightarrow} \begin{matrix} 1 & 0 & 0 & 0 \end{matrix} \begin{pmatrix} 0 & x_{11} & x_{13} & x_{14} \\ & 0 & x_{23} & x_{24} \\ & & \boxed{x_{33}} & x_{34} \\ & & & \boxed{x_{44}} \end{pmatrix} \overset{\text{QR}}{\rightarrow} \begin{pmatrix} 1 & 0 & 0 & 0 \\ & x_{11} & x_{13} & x_{14} \\ & & \tilde{x}_{22} & \tilde{x}_{23} \\ & & & \tilde{x}_{33} \end{pmatrix}.$$

Note that the bottom right triangular submatrix of the resulting matrix is our desired R_{11}:

$$R_{11} = \begin{pmatrix} x_{11} & x_{13} & x_{14} \\ & \tilde{x}_{22} & \tilde{x}_{23} \\ & & \tilde{x}_{33} \end{pmatrix}.$$

Also note that this R_{11} is *identical* in the two implementations. For reasons that will become clear in the next section, it is convenient to leave behind a "1" in position (1,1), such that after the k-th step, the contents of the triarray is

$$\begin{pmatrix} I & 0 \\ 0 & R_{11}^{(k)} \end{pmatrix}. \tag{8}$$

3.4 Multi-column shifts

As we have seen in Section 2, the Chan/Foster Algorithm produces estimates of the singular vectors corresponding to the smallest singular values sequentially, one at a time. This is actually not the only way to perform the RRQRF algorithm. Bischof & Hansen [3] recently presented a block

algorithm, based on incremental condition estimation [2], that seeks to estimate several singular vectors associated with small singular values at a time (in case $n - r >$), and thus allows one to accelerate the procedure by simultaneously—in each step of the Chan/Foster Algorithm—shifting several columns to the back of the matrix. Although well suited for high-performance computers, this block algorithm is not suited for a systolic array implementation because of the need for "global communication" in the matrix. In addition, Bischof and Hansen's algorithm requires a small QRF of the block of null vectors in each step of the algorithm, thus preventing a simple and efficient systolic realization.

There is, however, another possibility for simultaneously shifting more than one column at each iteration. The idea is to shift not only the p-th column that corresponds to a large element in the estimated singular vector v_k, but to shift two or more columns corresponding to the large elements in v_k. Our simulations show that when the threshold ρ (*cf.* step 2b of the Chan/Foster Algorithm) is close to one, some benefit can actually be obtained, at least on the average.

For instance, let us assume we want to shift the two columns corresponding to indices p and q (with $q < p$), according to the following rule: either both $v_k(p) \geq \rho\|v\|_\infty$ and $v_k(q) \geq \rho\|v\|_\infty$, if there are such p and q, or else p and q are such that $v_k(p)$ and $v_k(q)$ are the two largest components of v_k. The two columns picked according to this rule are shifted to the right simultaneously until column p reaches the k-th position. The shift can be implemented by performing consecutive swaps of adjacent columns. The time needed for shifting two separate columns ends up being the same as for shifting a single column, namely $\max\{p, q\}$. Therefore, the time spent in column swaps can be almost halved.

Note that if we opt for a multi-column strategy, then we have to complete the permutations before actually starting to zero out the sub-diagonals, in order to avoid conflicts of operations. This means that no pipelining is possible with the subsequent step where R's triangular form is restored. In addition, extra hardware is needed because more processors are required on the diagonal and different operations have to be performed by them. Moreover, the time spent in performing the Givens rotations is not changed, thus degrading the actual reduction in computing overhead.

In conclusion, the idea of shifting more than one column at a time requires more complex

structure and timing, but can actually reduce the total execution time of the Algorithm whenever ρ is close to one.

4 Implementation details

We shall now give a more detailled discussion of the most important implementation details for our systolic RRQRF array.

4.1 Implementation of the power method

At stage h of the POWER ALGORITHM, one has computed the vector $v^{(h-1)}$, and next iteration then requires the two following backsolves:

$$w^{(h)} = R^{-H} v^{(h-1)} \tag{9}$$

and

$$v^{(h)} = R^{-1} w^{(h)}. \tag{10}$$

We emphasize that the inverses of the matrices R and R^H are not actually needed, since only solutions of linear equations by backsubstitution are required.

Due to the special structure of R, and the fact that our algorithm forces its leading submatrix to be the identity matrix such that the matrix R stored in the array always has full rank, *cf.* Eq. (8), both substeps (9) and (10) can be performed on the *same* triangular array used for the QR decomposition, where the entries of R, r_{ij}, are frozen in the corresponding PE's [39]. This approach is more suited for a systolic array implementation than an algorithm that works with vectors of reduced length, since the size of the array remains fixed throughout the procedure. Of course, the starting vector $v^{(0)}$ must have zeros in its first $n-k$ components, in order to cope with this "artificial" R whose leading submatrix is the identity of order $n-k$. Observe that if we wish to always pick the same starting vector $v^{(0)}$ for all values of k, from n down to the numerical rank r, the only reasonable and safe choice is $v^{(0)} = (0, \ldots, 0, 1)^T$.

Consider the first substep (9). If $v^{(h-1)}$ is input from the top of the array in skewed form, then $w^{(h)}$ can be obtained at the right edge as an output. The operations of the PE's are described in

Table 1 in the next subsection.

Now two different approaches are possible for the second substep (10), according to the available hardware.

Method #1. One possible strategy, shown in Fig. 2, is to make use of an extra column of n PE's at the right edge of the array (denoted by a shaded rectangle in the figure), in order to store the computed components of $w^{(h)}$ and evaluate the vector $v^{(h)}$. The latter can be done by feeding the array with minus the identity matrix, $-I$. The PE's, including those in the extra column, perform the same operations as in the first substep. For more details about this strategy, see [39].

Method #2. Another way to compute $v^{(h)}$ from $w^{(h)}$ is to exploit the fact that the linear equations in (10) have the same *structure* as in step (9). As an example, consider the case $n = 3$:

First substep

$$\begin{aligned} w_1 &= \frac{1}{r_{11}} v_1 \\ w_2 &= \frac{1}{r_{22}} (v_2 - r_{12} w_1) \\ w_3 &= \frac{1}{r_{33}} (v_3 - r_{13} w_1 - r_{23} w_2). \end{aligned}$$

Second substep

$$\begin{aligned} v_3 &= \frac{1}{r_{33}} w_3 \\ v_2 &= \frac{1}{r_{22}} (w_2 - r_{23} v_3) \\ v_1 &= \frac{1}{r_{11}} (w_1 - r_{12} v_2 - r_{13} v_3). \end{aligned}$$

We see that if the links are all bidirectional, then a very simple way to implement (10) is to re-input $w^{(h)}$ from the right, reverse all the inputs and outputs of all the PE's, and obtain $v^{(h)}$ at the top of the array. Apart from the rescaling, the vector $v^{(h)}$ is now in place for the next iteration, *cf.* Fig. 2. No extra column of PE's is needed in this method.

In both cases, normalization or rescaling[1] is required between each iteration. This can be done by an external processor. This processor can also be in charge of finding a large component in

[1] By *rescaling* we mean here "multiplication of the vector's components by a pre-defined factor," where the factor need not be related to the vector's norm. *Normalization* is the particular case of rescaling by the reciprocal of the norm of the vector.

absolute value of the vector v_k, which is needed for the next step of the Algorithm. After the completion of the POWER ALGORITHM, the normalized $v^{(h)}$—augmented at the bottom with $n-k$ zeros—can be stored in an external memory, stacked with the previously computed singular vectors and thus forming the null-space matrix W, *cf.* Eq. (7).

Note that if rescaling is used in Method #1—instead of normalization—then a further speed-up of the procedure can be obtained, since one does not need to wait until the last component of the vector $v^{(h)}$ is computed to start the next iteration. The norm of $v^{(h)}$ is not evaluated at every iteration, but only after the final one, in order to compute a good estimate for the singular value. As for the choice of the scaling factor, one possibility could be to just use $v^{(h)}(1)$, the first element of $v^{(h)}$, since it can never be zero.

One more consideration can be made at this point. Assume that the starting vector $v^{(0)}$ for the Power Algorithm is chosen so that $v^{(0)} = (0, \ldots 0, 1)^T$ and $v^{(0)}(n) = 1$. Also, assume that *only one* iteration is required. In this case, $w^{(1)}$ will always assume the same form (namely, $w^{(1)} = r_{nn}^{-1} v^{(0)}$) and there is no need to compute it. Therefore, in Method #1, the identity matrix can be immediately fed into the array and the output from the (n,n)-th PE, being proportional to the true $v^{(1)}$, can be directly used to estimate the value of p. In this way, the extra column of PE's can be completely avoided. Choosing to perform only one power iteration may be considered somewhat risky, and must depend on the accuracy required by the particular application; but simulation shows that, at least with $\rho = 0.1$, one iteration is actually enough for many practical cases.

4.2 Implementation of the shift and rotations

The right shift of the first $(p-1)$ columns of R can be obtained by sending a control word to the input of the array (the top row). This can be done by the external processor, once it has determined the value of p. One possibility is to present a word with ones corresponding to columns 1 through p and zeros elsewhere. This word is propagated downwards unchanged through all the cells. When a cell sees a one at its input, then it makes its contents available to its right side neighboring cell. One of the outputs for the rotation parameters (say s) can be used for this purpose. If the cell is on the diagonal, then the value of r_{ii} stored in it has to be replaced by zero. If the cell is in the

interior of the array, the stored r_{ij} has to be replaced by the contents of the left side neighbor, which is now available at one of its inputs (corresponding to the same rotation parameter). A cell that receives a zero at its input does not need to perform any operations at all.

As we have mentioned before, the problem to find the actual value of p is left to the external processor. Observe that p need not correspond to the component of v_k having maximum absolute value. The algorithm performs almost as well if we choose an element $v_k(p)$ such that the ratio $v_k(p)/\|v_k\|_\infty$ is above a given threshold $\rho < 1$. A further desirable consequence of choosing $\rho < 1$ is that, in so doing, we can somewhat reduce the number of power iterations. As a matter of fact, after just one or two iterations of the power method the vector $v^{(h)}$ starts to resemble the true singular vector, in the sense that the components which will eventually dominate become "larger" while the other components tend to decrease. In other words, whenever the largest component of the estimated singular vector is required ($\rho = 1$), one has to iterate a sufficient number of times, but if only a "large" component is sought ($\rho < 1$), then the number of power iterates can be significantly lower. The number of power iterates can actually be chosen concomitantly with the value of the threshold ρ, on the basis of these considerations.

As explained in §3, the column shift generates a subdiagonal which needs to be annihilated via Givens rotations. This is simply obtained by switching back to the QR mode and inputting a row of a one followed by all zeros. The operations performed by the cells are therefore the same as during the preliminary QR step.

Table 1 summarizes the three different tasks performed by the processing elements of the systolic array.

5 Performance analysis

In this section we analyze in details the execution time of our proposed systolic array, and we compare it with current systolic arrays for computation of the SVD.

Table 2 summarizes the duration of each phase of the RRQRF procedure as well as the total execution time of the algorithm. The first entry of the table, denoted "QR", shows how many time slots (clock ticks) are required for the initial QR factorization of the $m \times n$ matrix A. Since the

Cell Type	QR	Sing. val. & vec.	Shift
Diagonal Cell	if $x_{in} = 0$, then $c \leftarrow 1; s \leftarrow 0$, else $r' \leftarrow \sqrt{r^2 + x_{in}^2}$; $c \leftarrow r/r'; s \leftarrow x_{in}/r'$; $r \leftarrow r'$ end	$s \leftarrow x_{in}/r$	if $x_{in} = 1$, then $s_{out} \leftarrow r; r \leftarrow 0$ end
Inner Cell	$x_{out} \leftarrow cx_{in} - sr$; $r \leftarrow sx_{in} + cr$	$x_{out} \leftarrow x_{in} - sr$	if $x_{in} = 1$, then $s_{out} \leftarrow r; r \leftarrow s_{in}$ end

Table 1: Operations of the processing elements (cells) in the different phases.

rows of the matrix are input one by one in a skewed fashion, *cf.* Fig. 1, one needs m time slots for a complete column to enter the array and n slots for a complete row. The complete $m \times n$ matrix A is passed through the whole array in a total of $m + 2n - 2$ time units, which is the time needed for the processing elements to settle to the final value.

The next row of the table shows the number of time slots required to estimate the smallest singular value and corresponding singular vector. Here, N_I represents the number of iterations required for this purpose. According to the considerations given previously in Section 3, N_I can be as small as 1 or 2. If Method #1 is chosen, $2n - 1$ clock ticks are needed to compute $w^{(h)}$ given $v^{(h-1)}$, and an extra $n + 1$ clock ticks are required to complete the iteration. On the other hand, if one chooses Method #2 then $w^{(h)}$ is obtained from $v^{(h-1)}$ in $2n - 1$ time units and the same amount of time is required to produce $v^{(h)}$.

Before starting the column shift one needs to determine the location p of the maximum component of the singular vector. Once this is accomplished, a control word has to be generated in order to shift columns 1 through $(p - 1)$ to the right and to fill the empty cells with zeros. In Table 2,

the time due to the determination of p and the generation of the control word is denoted by τ. Since it is required to scan all n components of the estimated singular vector, usually $\tau = O(n)$ for sequential computers, while $\tau = O(\log n)$ if the search is done in parallel. The number of clock ticks required for the column shift operation is therefore given by τ plus the time needed for the control word to be propagated along the array ($\simeq 2n$). In total, $\simeq 2n + \tau$ time slots are required for this phase.

Finally, consider the subsequent QR step which zeroes out the subdiagonal elements generated by the column shifts, and restores R into its triangular form. This step is simply obtained by inputting a row vector consisting of a one followed by all zeros, *cf.* §3. This operation requires a total of $2n - 1$ time slots to complete. Observe that in this analysis the time required to feed back the data via the feed back lines has been neglected; whether this is a reasonable assumption depends on the actual realization.

In order to evaluate the total time, we must consider that the operations can be pipelined. At the beginning of the procedure, the initial vector $v^{(0)}$ can be input immediately after the matrix A, thus completing the N_I iterations required by the singular vector estimation in either $m + 3nN_I$ or $m + 2(2n-1)N_I$ time steps, according to whether Method #1 or Method #2 is used. If $r = n$, the algorithm stops here. If A is not full rank, then the new leading triangular submatrix has to be computed. $R_{11}^{(n-1)}$ will be stored in the bottom left part of the array after the determination of p, the shift of the first $(p-1)$ columns and the zeroing of the subdiagonal, namely after $m + 3nN_I + 2n + \tau$ (respectively, $m + 2(2n-1)N_I + 2n + \tau$) time slots.

From this point on, estimation of the smallest singular value, column shift and restoration to triangular form can be pipelined. Any succeeding iteration of the RRQRF algorithm will therefore need $3nN_I + \tau + 2$ (resp., $2(2n-1)N_I + \tau + 2$) time units to complete (notice that p does not appear). In total, there will be $(n-r)$ complete iterations plus an additional Power Algorithm step to check that the next estimated singular value is actually above threshold. Thus, the total time for computing the RRQR factorization becomes $m + 3n(n-r+1)N_I + (\tau+2)(n-r)$ for Method #1 and $m + 2(2n-1)(n-r+1)N_I + (\tau+2)(n-r)$ for Method #2. In Table 2, we give these times together with the simpler case in which $m = n$ and $\tau = n$.

Operation	Method #1	Method #2
Init. QR	$2n+m-2$	$2n+m-2$
Sing. vector	$3nN_I$	$2(2n-1)N_I$
Column shift	$2n+\tau$	$2n+\tau$
Subs. QR	$2n-1$	$2n-1$
Total	$m+3n(n-r+1)N_I$ $+(\tau+2)(n-r)$	$m+2(2n-1)(n-r+1)N_I$ $+(\tau+2)(n-r)$
(if $m=n, \tau=n$)	$\sim(3N_I+1)n(n-r)$	$\sim(4N_I+1)n(n-r)$

Table 2: Duration of each phase and total execution time (N_I: number of iterations, r: rank, m, n: matrix dimensions, $\tau = O(n)$ is the delay needed to determine p: see paper).

Table 3 gives a rough description of the hardware needed by the systolic implementations of the RRQRF algorithm considered in this paper. Method #1 requires n additional PE's in order to store the vector $w^{(h)}$ in each iteration of the Power Algorithm and to compute the vector $v^{(h)}$. This vector has to be used for the next iteration of the Power Algorithm, thus requiring feedback wires. An external processor is needed to compute p and produce the required control word, which has to be stored in n buffers on top of the array.

In Method #2 feedback lines are essentially traded for bidirectional links and PE's that can switch inputs with outputs at given times. $2n$ buffers are needed to store the vector $w^{(h)}$ and the control word. Again, an external processor evaluates p.

Both methods require control signals to run across the array in order to scan the different stages of the procedure and define the sets of operations to be performed by the different processing cells (see Table 1).

From Tables 2 and 3, it is apparent that Method #1 actually requires a shorter execution time than Method #2, at the expense of a higher number of processing elements. The different structure requirements are difficult to compare, without additional information on technology and available

Method #1	Method #2
$n(n+3)/2$ PE's	$n(n+1)/2$ PE's
n buffers	$2n$ buffers
control logic	control logic
feedback lines	bidirectional links

Table 3: Hardware requirements.

hardware.

For comparison's sake, recall that implementations of Jacobi-type algorithms for the singular value decomposition of an $n \times n$ matrix [37, 7, 55] require aproximately $n^2/4$ processing elements and an execution time $O(N_S n)$, where N_S is the number of "sweeps" required for a good estimate of the singular values. The number of sweeps cannot be predicted with certainty, but there are heuristic arguments [7] for N_S to be of the order of $\log n$. However, as pointed out in [28], the null-vectors produced by Jacobi algorithms are not accurate unless either an additional refinement step is used or the rotations are accumulated in a separate array. Hence, the constant in the computational complexity $O(N_S n)$ is somewhat higher than that for the RRQRF algorithm. Moreover, the processing elements and the interconnection pattern for the SVD arrays have a complexity similar to those in the alternative, more complex, structure discussed in the Appendix.

Liu, Hsieh and Yao [36] recently proposed two alternative multistage systolic arrays for computing the SVD of symmetric matrices, based on the QR algorithm. The triangular array they describe requires $n(n+1)/2$ processing elements, n feedback lines, and a circular multiplexer. The execution time is $O((3N_S + 2)n)$. The second array proposed in [36] is a square array, thus requiring n^2 processing cells, with $O(n^2)$ delay elements, n feedback lines, and n buffers. This array (interconnection pattern included) is more complex, but allows greater numerical stability than the triangular one. The execution time, however, is somewhat larger than for the triangular array, being of the order of $O((10N_S + 2)n)$. It is found in [36] that N_S can vary greatly for different matrices, and that in the practical cases analyzed, N_S is proportional to n.

In summary, arrays based on Jacobi-like iterations tend to be more complex in terms of hardware and software. On the other hand, processors based on the QR algorithm are limited to symmetric matrices and complete the computations in a time which can be considerably longer than what is required by our array. Thus, when implemented correctly, our systolic RRQRF algorithm is certainly a competitor to the systolic arrays for SVD computation.

6 Conclusions

In this paper we have considered the Rank Revealing QRF Algorithm due to Foster and Chan, which allows to compute both the numerical rank of a matrix and its (approximate) null-space. One possible systolic implementation which makes use of the classical triangular array by Gentleman and Kung is analyzed. At the end of the operations, we obtain the numerical rank r, and the well-conditioned leading $r \times r$ submatrix R_{11} of R remains stored in the array. As a side-product of the algorithm, obtained at no extra expense, one also obtains a good approximate basis for the null-space of the matrix. This additional feature can be very useful (and sometimes is the most interesting aspect of the RRQRF Algorithm) in real-life applications. A possible different implementation is also briefly sketched in the Appendix.

We give a detailed discussion of the systolic implementation and the modifications that are necessary to the original Chan/Foster algorithm. We also point out that alternative strategies are possible, with trade-offs between computing time, storage, and complexity of the systolic array. Finally, we consider the performance of our proposed systolic RRQRF algorithm, and we compare it with the performance of systolic SVD algorithms that are also capable of computing the rank and null space of a matrix. We conclude that for a general matrix whose rank is comparable with the size of the matrix, our systolic RRQRF algorithm is competitive to the systolic SVD algorithms.

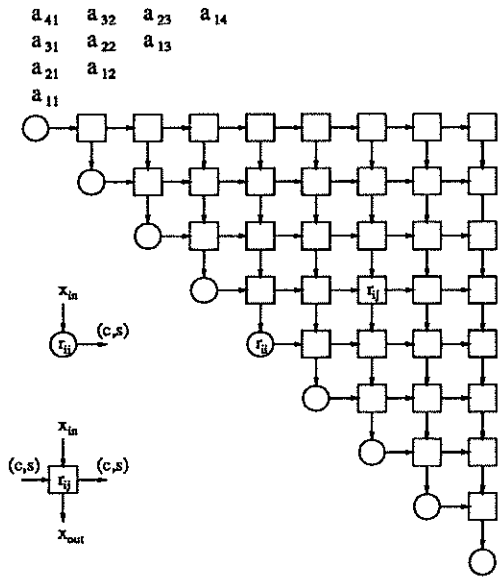

Figure 1: Gentleman and Kung's triangular array "triarray". The insert shows the two different processing elements.

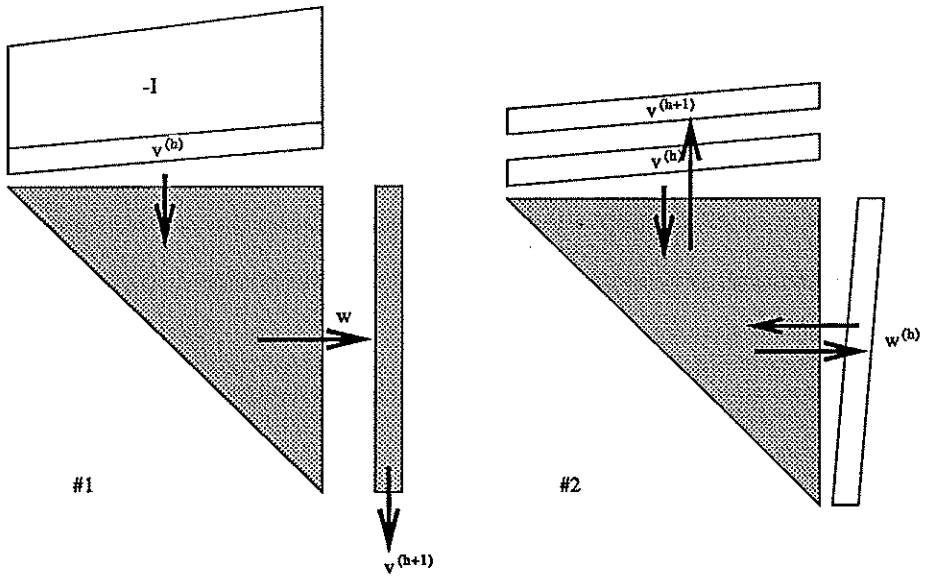

Figure 2: Methods #1 and #2 for implementing the Power Algortihm. Shaded areas denote procesing elements while white areas denote data. The figure does now show the hardware necessary to find a large element of the vector $v^{(h)}$.

Appendix

An alternative systolic RRQRF array

The main problem that arises when a systolic implementation of the RRQRF algorithm is sought is to efficiently use the parallel processing capability offered by systolic arrays. Consider the column shifting and the following subdiagonal zeroing by means of Givens rotations. The former task is obviously a column operation, wh reas the latter task involves the rows of the matrix. It would be desirable to somehow have both column shifting and Givens rotations happen together in parallel. If this were possible, we could store the whole matrix R in the array, without ever having to "eliminate" any of its columns. In this way, we would keep track of both submatrices R_{11} and R_{12}. With the array structure considered so far, it would not be impossible, but, at the least, slow and complicated to realize the aforementioned behavior. Besides, extra hardware would be needed on the main diagonal of the array. It is therefore conceivable that an alternative structure of the systolic array would enable us to achieve what we want in a simpler way.

A possible such structure could be a network of memory cells, each corresponding to an element of the matrix R, and a similar grid of processing elements, as described by Moonen [40]. In Fig. 3 the black circles represent memory cells, while the white circles and squares correspond to processing elements (boundary and internal cells, respectively).

Each processor has access to four storage elements and thus shares each of them with another processor. In addition to the connections between processor and memory cells, the PE's are connected with each other horizontally and vertically. Since neighboring PE's cannot be active at the same time (or else they would be working on the same data) each processor ends up being idle for 75% of the time. For this reason, four contiguous processors can be merged into a single "super-processor" (SP), as suggested in [40] and shown in the insert of Fig. 3. Each SP will have four storage cells of its own and will share the other eight with its six neighbors. If we proceed as described, the total number of SP's will be roughly eight times smaller than the number of matrix elements and they will all be working efficiently (*i.e.*, 100% of the time).

This array can perform both the QRF and power iteration steps in much the same way as

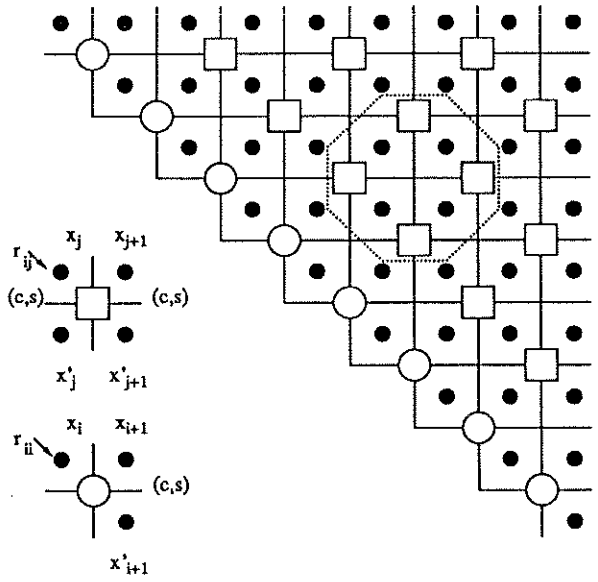

Figure 3: Alternative structure of a systolic RRQRF array.

the array described in §3, since all operations are exactly the same; only the data retrieval and movement change. What this structure actually helps to obtain is the combined column shift and row rotations. The columns are permuted two by two, starting from the p-th column, until the whole cycle is completed. At the *same time*, the subdiagonal is annihilated and the rotation parameters are propagated to the right. The diagonal SP's can in fact permute the matrix elements stored on their right hand side with those stored on their left hand side and simultaneously compute the suitable rotation parameters, without ever letting a subdiagonal show up. The off-diagonal SP's corresponding to the same columns will just permute the matrix elements in a similar fashion, while the SP's corresponding to the same rows will perform the rotation once they receive the parameters computed by the diagonal processor. Table 4 synthesizes the operations performed by the different cells in this alternative structure. We emphasize that the whole R matrix (and, in particular, the submatrices of interest R_{11} and R_{12}) is stored in this array at any given time.

Fig. 4 shows an example of the timing of the shift & rotate phase for a case in which $p = 2$. In the figure, "s" stands for "swap", "r" for "rotate", and "x" for "swap, compute the rotation

Cell Type	QR	Sing. val. & vec.	Shift & Rotate
Diagonal Cell	compute (c, s) $r_{ii} \leftarrow \sqrt{r_{ii}^2 + x_i^2}$ $x'_{i+1} \leftarrow cx_{i+1} - sr_{i,i+1}$ $r_{i,i+1} \leftarrow sx_{i+1} + cr_{i,i+1}$	$s \leftarrow x_i / r_{ii}$ $x'_{i+1} \leftarrow x_{i+1} - sr_{i,i+1}$	swap $(r_{i,i}, r_{i+1,i})$ and $(r_{i,i+1}, r_{i+1,i+1})$ then same as in QR phase

Inner Cell	apply the transformation to
	$(r_{i,j}, x_j)$ and $(r_{i,j+1}, x_{j+1})$
		$x'_j \leftarrow x_j - sr_{i,j}$
		$x'_{j+1} \leftarrow x_{j+1} - sr_{i,j+1}$
			if *rotate*, then
			rotate $(r_{i,j}, r_{i,j+1})$
			and $(r_{i+1,j}, r_{i+1,j+1})$;
			if *swap*, then
			swap $(r_{i,j}, r_{i+1,j})$
			and $(r_{i,j+1}, r_{i+1,j+1})$.

Table 4: Operations of the processing elements in the different phases for the alternative structure.

parameters and rotate." The number of "clock ticks" necessary for the completion of the whole phase, for a triangular array of dimension k, is $2(k-p) + \lfloor k/2 \rfloor - 1$. Without describing the details of how the single operations are scheduled on the array, we summarize in Table 5 the approximate duration of each phase and of the whole procedure (see also [40]; for the notation, *cf.* Table 2).

This alternative structure requires a low number of processors (about eight times smaller than the number of matrix elements in R), but they have to be capable of sharing memory cells with their neighbors. The execution time is longer than for the array based on the Gentleman-Kung processor, but in this way the whole final matrix R, and not just the submatrix R_{11}, will be available at the end of the procedure. The matrix W is computed in much the same way as for the array of §3, therefore in this respect the two structures are equivalent.

An *a priori* comparison between the architecture presented in this Appendix and the one considered in the paper is very difficult, if not impossible. Drawbacks and advantages are very much dependent on the specific application. Some of the applications listed in the Introduction explicitly need the R_{12} submatrix (*e.g.*, the truncated QR solution of a rank deficient least squares problem).

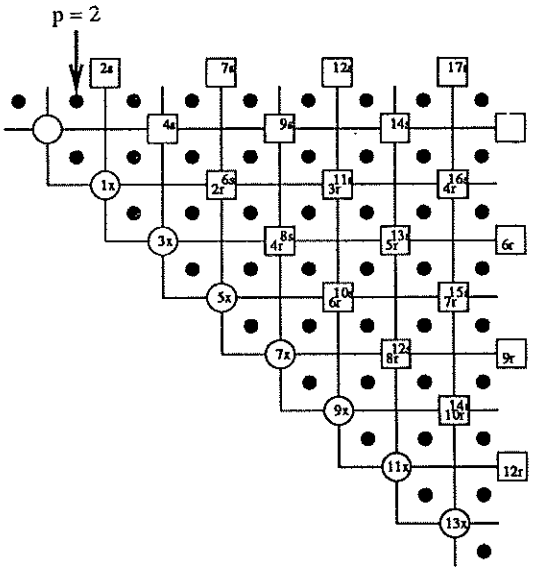

Figure 4: Example: timing of the shift & rotate phase.

Some do not. Besides, when no updating is required and all is needed is an estimation of rank and numerical null space (*e.g.*, DOA problems, curve fitting by total least squares), then it seems obvious that the architecture based on the Gentleman-Kung processor is more advantageous, since it has a lower computation time and less complex hardware (both cells and interconnections). If, on the other hand, updating is an issue, then the array described in this Appendix is probably better, since it allows one to compute the updated null space by the formula

$$W = \begin{pmatrix} -R_{11}^{-1} R_{12} \\ I \end{pmatrix},$$

as suggested in [5]. This computation could actually be performed using the same array. Notice, however, that even when the submatrix R_{12} is needed the architecture presented in the paper is not necessarily ruled out. As briefly mentioned in §3, R_{12} can be stored outside the main array. The choice of the actual architecture to adopt therefore involves computation time, complexity of the individual processors, interconnection pattern and also the availability of external storage and processing capabilities.

Operation	
QR	$2n + 4m - 4$
Sing. vector	$6n$
Column shift	$\sim 2.5n - 3 + \tau$
Total (if $m = n, \tau = n$)	$\sim (6N_I + 3.5)n(n - \tau)$

Table 5: Duration of each phase and total execution time for the alternative structure.

References

[1] E. Biglieri and K. Yao. "Some Properties of the SVD and their Application to Digital Signal Processing". *Signal Processing*, 18:277–289, 1989.

[2] C. H. Bischof. "Incremental Condition Estimation". *SIAM Journal on Matrix Analysis and Applications*, 11(2):312–322, April 1990.

[3] C. H. Bischof and P. C. Hansen. "A Block Algorithm for Computing Rank-Revealing QR Factorizations". Technical Report MCS-P251-0791, Mathematics and Computer Science Division, Argonne National Laboratory, August 1991.

[4] C. H. Bischof and P. C. Hansen. "Structure-Preserving and Rank-Revealing QR-Factorizations". *SIAM Journal on Scientific and Statistical Computing*, 12(6):1332–1350, November 1991.

[5] C. H. Bischof and G. M. Shroff. "On Updating Signal Subspaces". *IEEE Signal Processing*, 40(1):96–105, January 1992.

[6] A. W. Bojanczyk, R. P. Brent, and F. R. De Hoog. "QR Factorization of Toeplitz Matrices". *Numerische Mathematik*, 49:81–94, 1986.

[7] R. Brent and F. T. Luk. "The Solution of Singular-Value and Symmetric Eigenvalue Problems on Multiprocessor Arrays". *SIAM Journal on Scientific and Statistical Computing*, 6(1), January 1985.

[8] J. R. Bunch and C. P. Nielsen. "Updating the SVD". *Numerische Mathematik*, 31:111–129, 1978.

[9] P. A. Businger and G. H. Golub. "Linear Least Squares Solution by Householder Transformation". *Numerische Mathematik*, 7:269–276, 1965.

[10] T. F. Chan. "Rank-Revealing QR-Factorizations". *Linear Algebra and its Applications*, 88/89:67–82, 1987.

[11] T. F. Chan and P. C. Hansen. "Computing Truncated SVD Least Squares Solutions by Means of Rank Revealing QR-Factorizations". *SIAM Journal on Scientific and Statistical Computing*, 11:519–530, 1990.

[12] T. F. Chan and P. C. Hansen. "Low-Rank Revealing QR Factorization". Technical Report CAM 91-08, Department of Mathematics, UCLA, Los Angeles, California, May 1991.

[13] T. F. Chan and P. C. Hansen. "Some Applications of the Rank Revealing QR Factorization". *SIAM Journal on Scientific and Statistical Computations*, 13(3):727–741, May 1992.

[14] A. K. Cline, A. R. Conn, and C. F. Van Loan. "Generalizing the LINPACK Condition Estimator". In J. P. Hennart, editor, *Numerical Analysis; Lecture Notes in Mathematics*, volume 909, pages 73–83. Springer Verlag, 1982.

[15] J. D. Dixon. "Estimating Extremal Eigenvalues and Condition Numbers of Matrices". *SIAM Journal on Numerical Analysis*, 20:812–814, 1983.

[16] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *"LINPACK Users' Guide"*. SIAM, Philadelphia, 1979.

[17] R. Fierro and J. R. Bunch. "Colinearity and Total Least Squares". Technical report, Department of Mathematics, UC San Diego, 1991.

[18] L. V. Foster. "Rank and Null Space Calculations Using Matrix Decomposition without Column Interchanges". *Linear Algebra and its Applications*, 74:47–71, 1986.

[19] L. V. Foster. "The Probability of Large Diagonal Elements in the QR Factorization". *SIAM Journal on Scientific and Statistical Computing*, 11(3):531–544, May 1990.

[20] W. M. Gentleman and H. T. Kung. "Matrix Triangularisation by Systolic Arrays". In *Proceedings of SPIE, Real Time Signal Processing IV*, volume 298, pages 19–26, 1983.

[21] G. H. Golub. "Numerical Methods for Solving Linear Least Squares Problems". *Numerische Mathematik*, 7:206–216, 1965.

[22] G. H. Golub, V. Klema, and G. W. Stewart. "Rank Degeneracy and Least Squares Problems". Technical Report 456, Dept. of Computer Science, University of Maryland, 1976.

[23] G. H. Golub and C. F. Van Loan. "An Ånalysis of the TLS Problem". *SIAM Journal of Numerical Analysis*, 17:883–893, December 1980.

[24] G. H. Golub and C. F. Van Loan. *"Matrix Computations"*. John Hopkins, second edition, 1989.

[25] G. H. Golub and C. Reinsch. "Singular Value Decomposition and Least Squares Solution". *Numerische Mathematik*, 14:403–420, 1970.

[26] P. C. Hansen. "SVD — Theory and Applications". Technical Report NI-84-05, Technical University of Denmark, Lyngby, Denmark, 1984.

[27] P. C. Hansen. "The Truncated SVD As a Method for Regularization". *BIT*, 27:543–553, 1987.

[28] P. C. Hansen. "Reducing the Number of Sweeps in Hestenes' Method". In Ed. F. Deprettere, editor, *SVD and Signal Processing: Algorithms, Applications and Architectures*, pages 357–368. North-Holland, 1988.

[29] P. C. Hansen. "The 2-Norm of Random Matrices". *Journal of Computational and Applied Mathematics*, 23:117–120, 1988.

[30] N. J. Higham. "A Survey of Condition Number Estimation for Triangular Matrices". *SIAM Review*, 29(4), December 1987.

[31] S. Van Huffel and J. Vandewalle. "Subset Selection Using the Total Least Squares Approach in Collinearity Problems with Errors in the Variables". *Linear Algebra and Its Applications*, 88/89:695–714, 1987.

[32] S. Van Huffel and J. Vandewalle. *"The Total Least Squares Technique: Computation, Properties and Applications"*. Elsevier, 1988. E. F. Deprettere ed.

[33] S. Van Huffel and J. Vandewalle. *"The Total Least Squares Problem: Computational Aspects and Analysis"*. SIAM, Philadelphia, 1990.

[34] I. Karasalo. "A Criterion for Truncation of the QR Decomposition Algorithm for the Singular Least Squares Problem". *BIT*, 14:1–4, 1974.

[35] C. L. Lawson and R. J. Hanson. *"Solving Least Squares Problems"*. Prentice Hall, Englewood Cliffs, NJ, 1974.

[36] K. J. R. Liu, S. F. Hsieh, and K. Yao. *"Comparisons of Parallel SVD in VLSI Array Processors"*, pages 135–146. IEEE Press, H. S. Moscovitz, K. Yao, R. Jain Eds., 1991.

[37] F. T. Luk. "A Triangular Processor Array for Computing Singular Values". *Linear Algebra and Its Applications*, 77:259–273, 1986.

[38] T. A. Manteuffel. "An Interval Analysis Approach to Rank Determination in Linear Least Squares Problems". *SIAM Journal on Scientific and Statistical Computing*, 2:335–348, 1981.

[39] J. G. McWhirter and T. H. Shepherd. "An Efficient Systolic Array for MVDR Beamforming". In *Proceedings of the International Conference on Systolic Arrays*, pages 11–20, 1988.

[40] M. Moonen. *"Jacobi-Type Updating Algorithms for Signal Processing, System Identification and Control"*. PhD thesis, Katholieke Universiteit, Leuven, Belgium, November 1990.

[41] M. Moonen, B. De Moor, L. Vandenberghe, and J. Vandewalle. "On- and Off-Line Identification of Linear State Space Models". *International Journal of Control*, 49(1):219–232, 1989.

[42] B. Ottersten, M. Viberg, and T. Kailath. "Analysis of Algorithms for Sensor Arrays with Invariance Structure". In *Proceedings of ICASSP*, pages 2959–2962, 1990.

[43] S. Prasad and B. Chandna. "Direction-of-Arrival Estimation Using Rank Revealing QR Factorization". *IEEE Signal Processing*, 39(5):1224–1229, May 1990.

[44] A. Rahman and K. B. Yu. "TLS Approach for Frequency Estimation Using Linear Prediction". *IEEE Transactions on ASSP*, October 1987.

[45] S. K. Rao. *"Regular Iterative Algorithms and their Implementations on Processor Arrays"*. PhD thesis, Stanford University, Stanford, California, 1985.

[46] J. P. Reilly, W. G. Chen, and K. M. Wong. "A Fast QR-Based Array-Processing Algorithm". In *Proceedings of SPIE, Advanced Algorithms and Architectures for Signal Processing*, volume 975, pages 36–47, 1988.

[47] R. Roy and T. Kailath. "Invariance Techniques and High-Resolution Null Steering". In *SPIE Advanced Algorithms and Architectures for Signal Processing*, volume 975, pages 358–367, 1988.

[48] R. Roy and T. Kailath. "ESPRIT — Estimation of Signal Parameters via Rotational Invariance Techniques". *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-37:984–995, July 1989.

[49] R. H. Roy. *"ESPRIT — Estimation of Signal Parameters via Rotational Invariance Techniques"*. PhD thesis, Stanford University, Stanford, California, 1987.

[50] R. O. Schmidt. "Multiple Emitter Location and Signal Parameter Estimation". In *RADC Spectrum Estimation Workshop*, pages 243–258, Griffiths AFB, New York, 1979.

[51] R. O. Schmidt. *"A Signal Subspace Approach to Multiple Emitter Location and Spectral Estimation"*. PhD thesis, Stanford University, Stanford, California, 1981.

[52] G. W. Stewart. "An Updating Algorithm for Subspace Tracking". *IEEE Transactions on Signal Processing*, 40(6):1535–1541, 1992.

[53] P. Stoica and A. Nehorai. "Performance Comparison of Subspace Rotation and MUSIC Methods for Direction Estimation". *IEEE Transactions on Signal Processing*, 39(2), February 1991.

[54] M. Wax and T. Kailath. "Detection of Signals by Information Theoretic Criteria". *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-33:387–392, April 1985.

[55] B. Yang and J. F. Böhme. "Reducing the Computations af the SVD Array Given by Brent and Luk". In *Proceedings of SPIE, Advanced Algorithms and Architectures for Signal Processing IV*, volume 1152, 1989.