Parallel Algorithms for Toeplitz Systems

Richard P. Brent
Computer Sciences Laboratory
Australian National University
Canberra, Australia

Abstract

We describe some parallel algorithms for the solution of Toeplitz linear systems and Toeplitz least squares problems. First we consider the parallel implementation of the Bareiss algorithm (which is based on the classical Schur algorithm). The alternative Levinson algorithm is less suited to parallel implementation because it involves inner products.

The Bareiss algorithm computes the LU factorization of the Toeplitz matrix T without pivoting, so can be unstable. For this reason, and also for the application to least squares problems, it is natural to consider algorithms for the QR factorization of T. The first $O(n^2)$ serial algorithm for this problem was given by Sweet, but Sweet's algorithm seems difficult to implement in parallel. Also, despite the fact that it computes an orthogonal factorization of T, Sweet's algorithm can be numerically unstable.

We describe an algorithm of Bojanczyk, Brent and de Hoog for the QR factorization problem, and show that it is suitable for parallel implementation. This algorithm overcomes some (but not all) of the numerical difficulties of Sweet's algorithm. We briefly compare some other algorithms, such as the "lattice" algorithm of Cybenko and the "generalized Schur" algorithm of Chun, Kailath and Lev-Ari.

1. Introduction

A Toeplitz matrix $A = (a_{i,j})$ is one in which $a_{i,j}$ is a function of $j - i$. It is convenient to write a_{j-i} for $a_{i,j}$ and consider the $m+1$ by $n+1$ matrix $A = (a_{j-i})_{i=0,\ldots,m;j=0,\ldots,n}$. Toeplitz matrices often arise in digital signal processing, e.g. in linear prediction problems [17, 43] as well as in the discretization of certain integral equations and elsewhere in numerical analysis [13]. We are often interested in solving one or more linear systems

$$Ax = b \tag{1.1}$$

where A is square ($m = n$) and nonsingular, often symmetric positive definite; or in solving linear least squares problems

$$\min \|Ax - b\|_2 \tag{1.2}$$

Appeared in *Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms* (edited by G. H. Golub and P. Van Dooren), Springer-Verlag, 1991, 75–92. A preliminary version appeared in [9].

(where generally A is rectangular with $m > n$). We shall assume throughout that A, b etc are real and that A has rank $n+1$. We shall not consider block Toeplitz matrices, although many of the algorithms mentioned here are applicable (with appropriate modifications) to block Toeplitz problems.

Consider first the linear system (1.1). The "classical" Gaussian elimination or Cholesky decomposition algorithms [28, 57] involve $O(n^3)$ arithmetic operations. However, for over forty years "fast" algorithms involving only $O(n^2)$ operations have been known [1, 3, 20, 21, 22, 31, 41, 45, 47, 56, 58, 59].

The Toeplitz structure of A implies that the matrix-vector product Ax reduces to a convolution problem and can be computed via the fast Fourier transform in $O(n \log n)$ arithmetic operations. This observation suggests that $o(n^2)$ algorithms for (1.1) might exist, and indeed such algorithms were found about ten years ago by Brent et al [10], Bitmead & Anderson [4], and Morf [44]. More such algorithms have been found recently by de Hoog [29], Musicus [46], Kumar [36], and Ammar & Gragg [2]. These algorithms all require $O(n(\log n)^2)$ arithmetic operations and are termed "superfast" by Ammar & Gragg [2], although we prefer the description "asymptotically fast" as n may have to be very large before they are actually faster than the $O(n^2)$ algorithms (see Section 4). It is conceivable that (asymptotically) even faster algorithms exist.

In practice operation counts are not the only criterion. The speed of an implementation may depend on how readily the algorithm can be implemented on a pipelined or parallel architecture, and this may depend on whether inner products need to be computed or not. The overall cost depends on the difficulty of programming and debugging, so simple algorithms are generally preferable to complex ones. Numerical stability (or the lack of it) is an important consideration [13, 16, 18, 42].

Algorithms for solving (1.1) split into two main classes –

($\mathcal{A}$) those that compute A^{-1} or a factorization of A^{-1}, and
($\mathcal{B}$) those that compute a factorization of A.

In class ($\mathcal{A}$) we have for example the Levinson [41], Durbin [21] and Trench [54] algorithms which, in the symmetric case, are closely related to the classical Szegö recursions for polynomials orthogonal on the unit circle [2, 35, 53]. These algorithms generally require inner products and are stable if A is positive definite [13].

In class ($\mathcal{B}$) we have for example the algorithms of Bareiss [3], Brent, Kung & Luk [11, 12], and Kung and Hu [38], which are related to the classical algorithm of Schur [48] for the continued fraction representation of a holomorphic function in the unit disk [2, 32, 35].

The fast algorithms of Bitmead & Anderson [4] and Morf [44] are based on the concept of displacement rank [25, 26, 33, 34] and may be placed in class ($\mathcal{A}$), while those of Brent et al [10], de Hoog [29], Musicus [46], and Ammar & Gragg [2] are related to continued fractions and generalized Schur algorithms, so may be placed in class ($\mathcal{B}$).

Consider now the full rank linear least squares problem (1.2). The solution of (1.2) is

$$x = (A^T A)^{-1} A^T b,$$

but it is generally undesirable to compute x from this formula, for numerical reasons [28] and because $A^T A$ is not Toeplitz (although it does have low displacement rank). A better approach is to compute an "orthogonal factorization" of A, i.e. a factorization

$$A = QR \tag{1.3}$$

where Q is an $m+1$ by $n+1$ matrix with orthonormal columns and R is an $n+1$ by $n+1$ upper triangular matrix. Once (1.3) is known, it is easy to solve (1.2), using

$$Rx = Q^T b \tag{1.4}$$

or, to avoid explicit computation of Q, the "semi-normal equations"

$$R^T Rx = A^T b \tag{1.5}$$

As in the discussion of (1.1), there are two classes of algorithms – some algorithms find the inverse factorization $A\bar{R} = Q$ instead of $A = QR$. The algorithms which compute $\bar{R}$ are related to the "lattice" algorithm [17, 18, 19, 30, 43], while algorithms which compute R include those of Sweet [52], Bojanczyk, Brent & Kung [6, 42] and Chun, Kailath & Lev-Ari [14]. When considering rectangular matrices, "fast" means algorithms requiring $O(mn)$ operations, since the classical algorithms require $\Omega(mn^2)$ operations.

Clearly with exact arithmetic we have

$$\bar{R} = R^{-1},$$

but numerically there is a difference between algorithms which compute $\bar{R}$ and those which compute R. Cybenko [18] has analysed the classical lattice algorithm and he suggests that its good numerical properties may carry over to his algorithm [19] for the solution of (1.2). This is by no means certain, because of the use of non-Euclidean inner products. On the other hand, Sweet's algorithm can be unstable [6, 42], even though it computes the orthogonal decomposition (1.3). The algorithms of Bojanczyk, Brent & de Hoog [6] and Chun, Kailath & Lev-Ari [14] involve the downdating of a Cholesky factorization, and this is known to be a poorly conditioned problem [8, 51]. It is an open problem whether there is a fast ($O(mn)$), unconditionally stable algorithm for the computation of the QR factorization of a Toeplitz matrix. Such algorithms do exist when the Toeplitz matrix has a special structure [5, 18].

Another approach [27, 50] to the solution of (1.2) is to solve the $m+n+2$ by $m+n+2$ system

$$\begin{pmatrix} 0 & A^T \\ A & -I \end{pmatrix} \begin{pmatrix} x \\ r \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix} \tag{1.6}$$

to find both the solution vector x and the residual vector $r = Ax - b$. This can be done in $O(m(\log m)^2)$ operations, by generalizations of some of the algorithms mentioned above, for the matrix in (1.6) has low displacement rank. This approach has the advantage that iterative refinement [28, 57] may easily be applied to improve the accuracy of the solution, if this is necessary. Its disadvantage is its obvious inefficiency if $m \gg n$, unless the special structure is used to advantage [50].

In practice space requirements need to be considered. The problem (1.1) can be defined with $O(n)$ parameters, and its solution can be expressed with $O(n)$ parameters, but many algorithms require $\Omega(n^2)$ words of intermediate storage. Clearly this is necessary if the triangular factors of A or A^{-1} are stored explicitly, since these factors are not Toeplitz. However, with care and a little extra computation, it is possible to implement some of the algorithms mentioned above with only "linear", i.e. $O(n)$, storage. Similar remarks apply to algorithms for the solution of (1.2).

In Section 2 we describe a version of the Bareiss algorithm for solving the linear Toeplitz system (1.1), and show how it can be implemented efficiently in parallel, using only linear storage. We briefly compare some other fast algorithms for solving (1.1). In Section 3 we consider algorithms for solving (1.2), and in particular we describe the algorithm of Bojanczyk, Brent & de Hoog and its parallel implementation. Finally, in Section 4 we comment on some of the asymptotically fast algorithms for solving (1.1).

When considering parallel algorithm we shall concentrate on algorithms suitable for systolic arrays [37]. Because systolic arrays are such a restricted class of parallel machines, they can be simulated efficiently on most other classes of parallel machines (e.g. hypercubes or shared memory machines). Thus, our algorithms for systolic arrays are easily converted into efficient algorithms for these other classes of parallel machines (but not conversely).

2. A parallel Bareiss algorithm

Write (1.1) as

$$A^{(0)}x = b^{(0)}.$$

At iteration k of the Bareiss algorithm ($k = 1, \ldots, n$) we have two linear systems

$$A^{(\pm(k-1))}x = b^{(\pm(k-1))}$$

which are both equivalent to (1.1), and these are transformed into two further systems

$$A^{(\pm k)}x = b^{(\pm k)}.$$

The transformations are chosen to eliminate *diagonals*, rather than rows or columns. Because of the Toeplitz nature of A, each transformation requires only $O(n)$ operations. After k iterations, the matrix $A^{(+k)}$ has k zero diagonals above the main diagonal, and the matrix $A^{(-k)}$ has k zero diagonals below the main diagonal. Formally, if

$$Z_k = (\delta_{i-j+k})_{i=0,\ldots,n;j=0,\ldots,n}$$

is a shift matrix (premultiplication by Z_{+k} shifts k rows up with zero fill; premultiplication by Z_{-k} shifts k rows down with zero fill), then the transformations are defined by

$$A^{(-k)} = A^{(-(k-1))} - m_{-k}Z_{-k}A^{(k-1)},$$

$$b^{(-k)} = b^{(-(k-1))} - m_{-k}Z_{-k}b^{(k-1)},$$

$$A^{(+k)} = A^{(k-1)} - m_kZ_kA^{(-k)},$$

$$b^{(+k)} = b^{(k-1)} - m_kZ_kb^{(-k)}$$

for $k = 1, \ldots, n$, where

$$m_{-k} = a_{k,0}^{(-(k-1))}/a_0$$

and

$$m_{+k} = a_{0,k}^{(k-1)}/a_{n,n}^{(-k)}.$$

To ensure that the multipliers $m_{\pm k}$ are well-defined, we must assume that all the leading principal submatrices of A are nonsingular.

After n iterations we obtain an upper triangular system

$$A^{(-n)}x = b^{(-n)} \tag{2.1}$$

which may easily be solved in $O(n^2)$ operations. In fact, we have computed a triangular factorization of A, for it may be shown that $A = LU$, where $a_0 L = (A^{(n)})^{T2}$, $U = A^{(-n)}$, and "$T2$" denotes matrix transpose about the main antidiagonal [52]. Thus, the Bareiss algorithm has essentially the same numerical properties as Gaussian elimination without pivoting – it is numerically stable if A is positive definite or diagonally dominant (as is often the case in applications), but it is unstable in general. Provided real-time response is not required, it is possible to handle this potential instability by computing the residual vector $r = A\bar{x} - b$ (where $\bar{x}$ is the computed, possibly inaccurate solution), then applying iterative refinement [28, 57] or switching to a slower but more stable algorithm if $\|r\|$ is unacceptably large.

Note that L and U are not Toeplitz. The Toeplitz structure gradually disappears during the iterations of the Bareiss algorithm. In fact, all but the top k rows of $A^{(-k)}$ are Toeplitz, and all but the bottom k rows of $A^{(+k)}$ are Toeplitz. This seems to imply that $\Omega(n^2)$ storage space is required. However, it was shown by Brent et al [11, 12] that space $O(n)$ is sufficient to solve (2.1), as we can generate the rows of $A^{(-n)}$ as required for the backsubstitution process by running the Bareiss algorithm backwards (using the stored multipliers $m_{\pm k}$) –

$$A^{(k-1)} = A^{(k)} + m_k Z_k A^{(-k)}$$

and

$$A^{(-(k-1))} = A^{(-k)} + m_{-k} Z_{-k} A^{(k-1)}$$

for $k = n, \ldots, 1$. For details and numerical examples, see Brent & Luk [12].

It is easy to convert the Bareiss algorithm into a "semisystolic" algorithm [40, 55] in which the multipliers $m_{\pm k}$ are broadcast to all processing elements in a linear array of $n+1$ systolic cells. By a standard technique [40, 55] this semisystolic algorithm can be converted into a true systolic algorithm in which no broadcasting is required, only nearest neighbour communication. For details, we again refer to Brent & Luk [12]. Here we merely note that one systolic processor needs to perform divisions (to compute the multipliers); the others need only perform "multiply and accumulate" operations of the form $\alpha \leftarrow \alpha + \beta \cdot \gamma$.

The systolic implementation of the Bareiss algorithm uses $n+1$ systolic cells and solves (1.1) in time $O(n)$ and space $O(n)$, i.e. constant space per systolic cell. On other parallel architectures such as transputer arrays, hypercubes, etc with p processors, we can simulate the systolic array in time $O(n^2/p)$, provided $p = O(n)$. Thus, the speedup is of order p and the efficiency is of order unity.

The algorithm described above does not take advantage of any symmetry. However, a symmetric version of the algorithm exists, and saves some time and communications overhead if A is symmetric [12, 38].

The algorithms in class ($\mathcal{A}$) generally require less arithmetic operations than the Bareiss algorithm (by a moderate constant factor), but they involve a sequence of $O(n)$ inner products which need to be performed sequentially. On a parallel machine with $p = O(n)$ processors, each inner product takes time $\Omega(n/p + \log p)$, so a parallel implementation of these algorithms will take time $\Omega(n^2/p + n \log p)$. In particular, if $p \sim n$, the time is $\Omega(n \log n)$, greater than the time $O(n)$ for the Bareiss algorithm.

3. Fast Toeplitz QR Factorization

Sweet [52] gave the first fast ($O(mn)$) algorithm for the orthogonal factorization (1.3) of a Toeplitz matrix. The classical Givens and Householder algorithms [28, 39, 50, 57] form Q as a product of elementary orthogonal matrices, so the computed matrix Q is guaranteed to be very close to orthogonal. Unfortunately, this is not the case for Sweet's algorithm – it turns out that the computed Q can deviate significantly from orthogonality because of the effect of rounding errors during the computation. (Similar remarks apply to algorithms which compute R or R^{-1} and then find Q from $Q = AR^{-1}$ – this approach is unsatisfactory if R is poorly conditioned.) Luk & Qiao [42] show (for the case $m = n$) that Sweet's algorithm is unstable if the matrix formed by deleting the first row and last column of A is poorly conditioned. We recommend the numerical examples given by Luk & Qiao to anyone interested in understanding the numerical behaviour of the Sweet, Bareiss and Trench algorithms.

Bojanczyk, Brent & de Hoog [6] (BBH for short) discovered a different Toeplitz QR factorization algorithm while trying to understand the reason for numerical instability in Sweet's algorithm. The key idea is to obtain a recursion for the rows of R (and, optionally, for the columns of Q) from the shift-invariance property of Toeplitz matrices. We partition the Toeplitz matrix A in two ways –

$$A = \begin{pmatrix} a_0 & y^T \\ z & A_{-1} \end{pmatrix} = \begin{pmatrix} A_{-1} & \bar{y} \\ \bar{z}^T & a_{n-m} \end{pmatrix}$$

and partition R in two corresponding ways –

$$R = \begin{pmatrix} r_{0,0} & u^T \\ 0 & R_b \end{pmatrix} = \begin{pmatrix} R_t & \bar{u} \\ 0 & r_{n,n} \end{pmatrix},$$

where u, $\bar{u}$, y, $\bar{y}$, z and $\bar{z}$ are column vectors, A_{-1} is the principal (top left or bottom right – they are identical) m by n submatrix of A, R_t is the top left principal n by n submatrix of R, and R_b is the bottom right principal n by n submatrix of R. Using the relation $R^T R = A^T A$ we obtain

$$r_{0,0} = (a_0^2 + z^T z)^{1/2}, \tag{3.1}$$

$$r_{0,0} u = a_0 y + A_{-1}^T z, \tag{3.2}$$

and

$$R_b^T R_b = R_t^T R_t + y y^T - \bar{z}\bar{z}^T - u u^T. \tag{3.3}$$

Relations (3.1) and (3.2) define the first row of R. Relation (3.3) shows that R_b differs from R_t by a matrix of rank 3. In fact, R_b can be obtained from R_t by a rank-1 update followed by two rank-1 downdates. This allows us to calculate the kth row of R_b from the first k rows of R_t. However, the kth row of R_b defines the $(k+1)$st row of R_t. Thus, we have a recursion for calculating the rows of R, starting from the first. It is possible to obtain a similar recursion for the columns of Q, although this is not necessary if (1.5) is used to solve the least squares problem (1.2).

Despite their similar derivations, the BBH algorithm and Sweet's algorithm are different. The operation count for the BBH algorithm is slightly lower than for Sweet's algorithm [6]. While Sweet's algorithm appears difficult to implement in parallel, the BBH algorithm can be implemented on a systolic array and runs in time $O(m)$ with $n+1$ processors [7]. Also, because the rows of R can be generated in reverse order by running the algorithm backwards (as for the Bareiss algorithm described in Section 2), the linear least squares problem (1.2) can be solved with space $O(m)$. Finally, the numerical properties of the BBH algorithm and Sweet's algorithm differ – numerical experiments suggest that the BBH algorithm is never much less accurate that Sweet's algorithm, and often much more accurate.

Updating Cholesky factors after a (positive) rank-1 correction can be done in a numerically stable way using plane rotations (i.e. circular transformations) [24]. However, it is known that downdating of Cholesky factors is numerically dangerous [8, 24, 51]. This is to be expected because the (real) Cholesky factorization of R^TR-vv^T only exists if $R^TR - vv^T$ has no negative eigenvalues. Thus, we would anticipate numerical difficulties if $R_b^TR_b$ is poorly conditioned. We conjecture that the BBH algorithm is *weakly stable* in the sense that R is computed accurately provided that $R^TR \ (= A^TA)$ is well-conditioned. Similarly, we conjecture that the BBH algorithm combined with the seminormal equations (1.5) gives a weakly stable algorithm for the solution of the linear least squares problem (1.2) so long as $\min \|Ax-b\|_2 \ll \|b\|_2$.

Recently Chun, Kailath & Lev-Ari [14] (CKL for short) presented a Toeplitz QR factorization algorithm (or family of algorithms) based on the fact that R is a Cholesky factor of the matrix A^TA, and this matrix has low displacement rank. They were able to interpret the BBH algorithm in this context, and show that it is closely related to the CKL algorithm. The algorithms have similar operation counts – we shall not be specific because the operation counts depend on details such as whether "fast" Givens transformations [23] are used. Because the algorithms are so closely related, it is not surprising that the CKL algorithm has a nice parallel implementation similar to that of the BBH algorithm.

It might be supposed that the CKL algorithm would have better numerical properties than the BBH algorithm because the CKL algorithm uses less hyperbolic transformations (and correspondingly more circular transformations) than the BBH algorithm. However, this is not necessarily the case. Equation (3.3) suggests that the critical factor, at least for the BBH algorithm, is the condition of $R_b^TR_b$, which is dependent on the original Toeplitz matrix A but not on the number of hyperbolic transformations performed by the algorithm. There is no need to invoke several hyperbolic transformations as a cause of instability when a single one is sufficient. As an analogy, we mention that the presence of just one large multiplier during Gaussian elimination without pivoting is sufficient to cause instability.

Cybenko [19] has recently suggested a quite different Toeplitz orthogonalization algorithm. His algorithm finds $\bar{R}$ and Q such that $A\bar{R} = Q$, and is based on a generalization of the "lattice" algorithm [17, 30, 43], which solves the same problem when A is restricted to have a special structure. Because Cybenko's algorithm uses orthogonalization with respect to a non-Euclidean inner product, its numerical properties are uncertain.

The comparison of Cybenko's algorithm with the BBH and CKL algorithms is analogous to the comparison of Levinson's algorithm with the Bareiss algorithm. Cybenko's algorithm uses less arithmetic operations and has different numerical properties, but it is less well suited to parallel implementation because it requires the evaluation of inner products.

4. Asymptotically fast algorithms

The asymptotically fast algorithms mentioned in Section 1 require $O(n(\log n)^2)$ operations to solve the Toeplitz linear system (1.1). The theory of these algorithms is fascinating as it exhibits connections between many apparently unrelated topics – Szegö polynomials, Schur's algorithm, Padé approximants, the Euclidean algorithm, etc. Space does not permit a discussion of these topics, so we refer to the literature [2, 10, 32, 35].

Sexton et al [49] evaluated the constant hidden in the "O" notation for an asymptotically fast algorithm based on the concept of displacement rank [4], and found that the constant was so large that the algorithm was impractical. More recent asymptotically fast algorithms, such as that of de Hoog [29], appear to have a smaller constant. Ammar & Gragg [2] have made a thorough attempt to minimize the constant and claim that their generalized Schur algorithm should be faster than Levinson's algorithm for $n > 256$ (approximately) when applied to a positive definite system. The comparison here is with the classical Levinson algorithm rather than with the slightly faster "split Levinson" algorithm.

All the asymptotically fast algorithms involve a "doubling step" in which the problem of size n is split into two subproblems of size approximately $n/2$, the results of which can be combined by some operations equivalent to convolution. Thus the time $T(n)$ required satisfies

$$T(n) = 2T(n/2) + C(n), \tag{4.1}$$

where $C(n)$ is the time required for a convolution of size n. Using the fast Fourier transform (or other fast convolution algorithm) gives $C(n) = O(n \log n)$ and thus (4.1) implies $T(n) = O(n(\log n)^2)$. Of course, the constant factor here depends on the constant for the convolution algorithm.

When we attempt to implement the asymptotically fast algorithms on a parallel machine, (4.1) still holds because the two subproblems are dependent – the second can not be performed until the first is complete. On a linear systolic array [37] we have $C(n) = O(n)$, so (4.1) implies that $T(n) = O(n \log n)$. Even on a more general parallel machine such as a hypercube, with $C(n) = O((\log n)^\alpha)$ for some positive constant α, we only get $T(n) = O(n)$, which is disappointing as the same result can be obtained much more easily by the parallel Bareiss algorithm (Section 2). On a hypercube with enough processors it is certainly possible to obtain $T(n) = O((\log n)^2)$, but we only know how to do this by ignoring the Toeplitz structure and using a fast but very inefficient algorithm such as that of Csanky [15], which requires $O(n^{3.5})$ processors.

5. Concluding remarks and open problems

Toeplitz systems are of interest to mathematicians because they display connections between various apparently unrelated areas of mathematics. They are of interest to numerical analysts and engineers because they arise in many applications and need to be solved quickly and accurately. A large literature exists, and many interesting algorithms have been developed, but there is still room for progress. We mention a few interesting problem areas in which progress on algorithms might be made –

1. The development of fast, provably stable algorithms for computing the QR decomposition of a Toeplitz matrix A and solving the linear least squares problem (1.2).
2. The discovery of "asymptotically fast" algorithms which are actually fast (compared to $O(n^2)$ algorithms) for moderate values of n.
3. The development of good parallel algorithms which solve the Toeplitz system (1.1) in time $O((\log n)^2)$ on a hypercube, using only a moderate number of processors (say $O(n^2)$).
4. Generalization of 1-3 above to algorithms with provably good numerical properties for low displacement rank matrices.
5. The discovery of good algorithms for other problems involving Toeplitz and low displacement rank matrices, e.g. eigenvalue and inverse eigenvalue problems.

References

1. H. A. Ahmed, J-M. Delsome and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing", *IEEE Transactions on Computers* 15 (1982), 65-82.
2. G. S. Ammar and W. B. Gragg, "Superfast solution of real positive definite Toeplitz systems", *SIAM J. Matrix Anal. Appl.* 9 (1988), 61-76.
3. E. H. Bareiss, "Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices", *Numerische Mathematik* 13 (1969), 404-424.
4. R. R. Bitmead and B. D. O. Anderson, "Asymptotically fast solution of Toeplitz and related systems of linear equations", *J. Linear Algebra Appl.* 34 (1980), 103-116.
5. A. W. Bojanczyk and R. P. Brent, "Parallel solution of certain Toeplitz least squares problems", *J. Linear Algebra Appl.* 77 (1986), 43-60.
6. A. W. Bojanczyk, R. P. Brent and F. R. de Hoog, "QR factorization of Toeplitz matrices", *Numerische Mathematik* 49 (1986), 81-94.
7. A. W. Bojanczyk, R. P. Brent and F. R. de Hoog, "Linearly connected arrays for Toeplitz least squares problems", Report CMA-R06-85, Centre for Mathematical Analysis, Australian National University, May 1985.
8. A. W. Bojanczyk, R. P. Brent, P. Van Dooren and F. R. de Hoog, "A note on downdating the Cholesky factorization", *SIAM J. Scientific and Statistical Computing* 8 (1987), 210-221.

9. R. P. Brent, “Old and new algorithms for Toeplitz systems”, *Proceedings SPIE, Volume 975, Advanced Algorithms and Architectures for Signal Processing III* (edited by Franklin T. Luk), Society of Photo-Optical Instrumentation Engineers, Bellingham, Washington, 1989, 2-9.
10. R. P. Brent, F. G. Gustavson and D. Y. Y. Yun, “Fast solution of Toeplitz systems of equations and computation of Padé approximants”, *J. Algorithms* 1 (1980), 259-295.
11. R. P. Brent, H. T. Kung and F. T. Luk, “Some linear-time algorithms for systolic arrays”, in *Information Processing 83* (edited by R.E.A. Mason), North-Holland, Amsterdam, 1983, 865-876.
12. R. P. Brent and F. T. Luk, “A systolic array for the linear-time solution of Toeplitz systems of equations”, *J. VLSI and Computer Systems* 1 (1983), 1-23.
13. J. Bunch, “Stability of methods for solving Toeplitz systems of equations”, *SIAM J. Scientific and Statistical Computing* 6 (1985), 349-364.
14. J. Chun, T. Kailath and H. Lev-Ari, “Fast parallel algorithms for QR and triangular factorization”, *SIAM J. Scientific and Statistical Computing* 8 (1987), 899-913.
15. L. Csanky, “Fast parallel matrix inversion algorithms”, *SIAM J. on Computing* 5 (1976), 618-623.
16. G. Cybenko, “The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations”, *SIAM J. Scientific and Statistical Computing* 1 (1980), 303-320.
17. G. Cybenko, “A general orthogonalization method with applications to time series analysis and signal processing”, *Mathematics of Computation* 40 (1983), 323-336.
18. G. Cybenko, “The numerical stability of lattice algorithms for least squares linear prediction problems”, *BIT* 24 (1984), 441-455.
19. G. Cybenko, “Fast Toeplitz orthogonalization using inner products”, *SIAM Jnl. Scientific and Statistical Computing* 8 (1987), 734-740.
20. P. Delsarte, Y. Genin and Y. Kamp, “A polynomial approach to the generalized Levinson algorithm based on the Toeplitz distance”, *IEEE Trans. Inform. Theory* IT-29 (1983), 268-278.
21. J. Durbin, “The fitting of time-series models”, *Rev. Inst. Internat. Statist.* 28 (1960), 233-244.
22. B. Friedlander, M. Morf, T. Kailath and L. Ljung, “New inversion formulas for matrices classified in terms of their distance from Toeplitz matrices”, *J. Linear Algebra Appl.* 27 (1979), 31-60.
23. M. Gentleman, “Least squares computations by Givens transformations without square roots”, *J. Inst. Math. Appl.* 12 (1973), 329-336.
24. P. E. Gill, G. H. Golub, W. Murray and M. A. Saunders, “Methods for modifying matrix factorizations”, *Mathematics of Computation* 28 (1974), 505-535.
25. I. C. Gohberg and I. A. Feldman, *Convolution Equations and Projection Methods for their Solution*, Translations of Mathematical Monographs, Vol. 41, Amer. Math. Soc., Providence, Rhode Island, 1974.
26. I. C. Gohberg and A. A. Semencul, “On the inversion of finite Toeplitz matrices and their continuous analogs”, *Mat. Issled.* 2 (1972), 201-233 (in Russian).
27. G. H. Golub, “Numerical methods for solving linear least squares problems”, *Numerische Mathematik* 7 (1965), 206-216.

28. G. H. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins Press, Baltimore, Maryland, 1983.
29. F. R. de Hoog, "A new algorithm for solving Toeplitz systems of equations", *J. Linear Algebra Appl.* 88/89 (1987), 122-138.
30. F. Itakura and S. Saito, "Digital filtering techniques for speech analysis and synthesis", *Proc. 7th Internat. Congr. Acoustics*, Budapest (1971), 261-264.
31. A. K. Jain, "An efficient algorithm for a large Toeplitz set of linear equations", *IEEE Trans. Acoust. Speech Sig. Proc.* 27 (1979), 612-615.
32. T. Kailath, "A theorem of I. Schur and its impact on modern signal processing", in *Schur methods in Operator Theory and Signal Processing*, edited by I. C. Gohberg, Birkhäuser-Verlag, Basel (1986), 9-30.
33. T. Kailath, S. Y. Kung and M. Morf, "Displacement ranks of matrices and linear equations", *J. Math. Appl.* 68 (1979), 395-407.
34. T. Kailath, B. Levy, L. Ljung and M. Morf, "The factorization and representation of operators in the algebra generated by Toeplitz operators", *SIAM J. Appl. Math.* 37 (1979), 467-484.
35. T. Kailath, A. Viera and M. Morf, "Inverses of Toeplitz operators, innovations, and orthogonal polynomials", *SIAM Review* 20 (1978), 106-119.
36. R. Kumar, "A fast algorithm for solving a Toeplitz system of equations", *IEEE Trans. Acoust. Speech and Signal Proc.* 33 (1985), 254-267.
37. H. T. Kung, "Why systolic architectures ?", *IEEE Computer* 15, 1 (1982), 37-46.
38. S. Y. Kung and Y. H. Hu, "A highly concurrent algorithm and pipelined architecture for solving Toeplitz systems", *IEEE Trans. Acoust. Speech and Signal Proc.* ASSP-31 (1983), 66-76.
39. C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.
40. C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems", *J. of VLSI and Computer Systems* 1 (1983), 41-67.
41. N. Levinson, "The Wiener RMS (root mean square) error criterion in filter design and prediction", *J. Math. Phys.* 25 (1947), 261-278.
42. F. T. Luk and S. Qiao, "A fast but unstable orthogonal triangularization technique for Toeplitz matrices", *J. Linear Algebra Appl.* 88/89 (1987), 495-506.
43. J. Makhoul, "Linear prediction: a tutorial review", *Proc. IEEE*, 63 (1975), 561-580.
44. M. Morf, "Doubling algorithms for Toeplitz and related equations", *Proc. IEEE Internat. Conf. on Acoustics, Speech and Signal Processing*, Denver, Colorado (April 1980), 954-959.
45. M. Morf and J-M. Delosme, "Matrix decompositions and inversions via elementary signature-orthogonal transformations", *Proc. Internat. Symposium on Mini- and Micro-computers in Control and Measurement*, San Francisco, California (May 1981).
46. B. R. Musicus, "Levinson and fast Choleski algorithms for Toeplitz and almost Toeplitz matrices", Report, Research Lab. of Electronics, MIT, Cambridge, Massachusetts, 1984.
47. J. Rissanen, "Solution of linear equations with Hankel and Toeplitz matrices", *Numerische Mathematik* 22 (1974), 361-366.
48. J. Schur, "Über Potenzreihen, die im Innern des Einheitskreises beschränkt sind", *J. Reine Angew. Math.* 147 (1917), 205-232.

49. H. Sexton, M. Shensa and J. Speiser, “Remarks on a displacement-rank inversion method for Toeplitz systems”, *J. Linear Algebra Appl.* 45 (1982), 127-130.
50. G. W. Stewart, *Introduction to Matrix Computations*, Academic Press, New York, 1973.
51. G. W. Stewart, “The effects of rounding error on an algorithm for downdating a Cholesky factorization”, *J. Inst. Math. Appl.* 23 (1979), 203-213.
52. D. R. Sweet, “Fast Toeplitz orthogonalization”, *Numer. Math.* 43 (1984), 1-21.
53. G. Szegö, *Orthogonal Polynomials*, third edition, AMS Colloquium Pub. vol. 23, Amer. Math. Soc. , Providence, Rhode Island (1967).
54. W. F. Trench, “An algorithm for the inversion of finite Toeplitz matrices”, *J. Soc. Indust. Appl. Math.* 12 (1964), 515-522.
55. J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1984.
56. G. A. Watson, “An algorithm for the inversion of block matrices of Toeplitz form”, *J. ACM* 20 (1973), 409-415.
57. J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
58. S. Zohar, “Toeplitz matrix inversion: the algorithm of W. F. Trench”, *J. ACM* 16 (1969), 592-601.
59. S. Zohar, “The solution of a Toeplitz set of linear equations”, *J. ACM* 21 (1974), 272-276.