Chapter 4

STABILITY OF FAST ALGORITHMS FOR STRUCTURED LINEAR SYSTEMS

Richard P. Brent

4.1 INTRODUCTION

This chapter surveys the numerical stability of some fast algorithms for solving systems of linear equations and linear least-squares problems with a low displacement rank structure. For example, the matrices involved may be Toeplitz or Hankel. We consider algorithms that incorporate pivoting without destroying the structure (cf. Sec. 1.13) and describe some recent results on the stability of these algorithms. We also compare these results with the corresponding stability results for the well-known algorithms of Schur–Bareiss and Levinson, and for algorithms based on the seminormal equations.

As is well known, the standard direct method for solving dense $n \times n$ systems of linear equations is Gaussian elimination with partial pivoting. The usual implementation requires arithmetic operations of order n^3.

In practice, linear systems often arise from some physical system and have a structure that is a consequence of the physical system. For example, time-invariant physical systems often give rise to *Toeplitz* systems of linear equations (Sec. 4.3.2). An $n \times n$ Toeplitz matrix is a dense matrix because it generally has n^2 nonzero elements. However, it is determined by only $O(n)$ parameters (in fact, by the $2n-1$ entries in its first row and column). Similar examples are the Hankel, Cauchy, Toeplitz-plus-Hankel, and Vandermonde matrices (see, e.g., Sec. 1.3 of this book, as well as [GV96] and [GKO95]).

When solving such a structured linear system it is possible to ignore the structure, and this may have advantages if standard software is available and n is not too large. However, if n is large or if many systems have to be solved, perhaps with real-time constraints (e.g., in radar and sonar applications), then it is desirable to take advantage of the structure. The primary advantage to be gained is that the time to solve a linear system is reduced by a factor of order n to $O(n^2)$. Storage requirements may also be reduced by a factor of order n, to $O(n)$.

Most works concerned with algorithms for structured linear systems concentrate on

the speed (usually measured in terms of the number of arithmetic operations required) and ignore questions of numerical accuracy. However, it is dangerous to use fast algorithms without considering their numerical properties. There is no point in obtaining an answer quickly if it is much less accurate than is justified by the data.

In this chapter we consider both the speed and the numerical properties of fast algorithms. Because there are many classes of structured matrices, and an ever-increasing number of fast algorithms, we cannot attempt to be comprehensive. Our aim is to introduce the reader to the subject, illustrate some of the main ideas, and provide pointers to the literature.

In this chapter, a "fast" algorithm will generally be one that requires $O(n^2)$ arithmetic operations, whereas a "slow" algorithm will be one that requires $O(n^3)$ arithmetic operations. Thus a fast algorithm should (in general) be faster than a slow algorithm if n is sufficiently large.

The subject of numerical stability and instability of fast algorithms is confused for several reasons:

1. Structured matrices can be very ill-conditioned [Tyr94b]. For example, the Hilbert matrix [Hil94], defined by $a_{ij} = 1/(i+j-1)$, is often used as an example of a very ill-conditioned matrix [FM67]. The Hilbert matrix is a Hankel matrix. Reversing the order of the rows gives a Toeplitz matrix. Even a stable numerical method cannot be expected to give an accurate solution when it is applied to a very ill-conditioned problem. Thus, when testing fast algorithms we must take the condition of the problem into account and not expect more than is reasonable.

2. The solution may be less sensitive to structured perturbations (i.e., perturbations that are physically plausible because they preserve the structure) than to general (unstructured) perturbations. The effect of rounding errors in methods that ignore the structure is generally equivalent to the introduction of unstructured perturbations. Ideally we should use a method that introduces only structured perturbations, but this property often does not hold or is difficult to prove, even for methods that take advantage of the structure to reduce the number of arithmetic operations.

3. The error bounds that can be proved are usually much weaker than those observed on "real" or "random" examples. Thus, methods that are observed to work well in practice cannot always be guaranteed, and it is hard to know if it is just the analysis that is weak or if the method fails in some rare cases. (A classical example of the latter phenomenon is given in Sec. 4.2.1.)

4. An algorithm may perform well on special classes of structured matrices, e.g., positive-definite matrices or Toeplitz matrices with positive reflection coefficients, but perform poorly or break down on broader classes of structured matrices.

4.1.1 Outline

Different authors have given different (and sometimes inconsistent) definitions of *stability* and *weak stability*. We follow Bunch [Bun85], [Bun87]. For completeness, our definitions are given in Sec. 4.2.

The concept of *displacement rank*, defined in Sec. 4.3 and also in Ch. 1 of this book, may be used to unify the discussion of many algorithms for structured matrices [KC94], [KKM79a], [KS95a]. It is well known that systems of n linear equations with a low displacement rank (e.g., Toeplitz or Hankel matrices) can be solved in $O(n^2)$ arithmetic op-

erations. Asymptotically faster algorithms with time bound $O(n \log^2 n)$ exist [BGY80] but are not considered here because their numerical properties are generally poor and the constant factors hidden in the "O" notation are large [AG88].

For positive-definite Toeplitz matrices, the first $O(n^2)$ algorithms were introduced by Kolmogorov [Kol41], Wiener [Wie49], and Levinson [Lev47]. These algorithms are related to recursions of Szegő [Sze39] for polynomials orthogonal on the unit circle. Another class of $O(n^2)$ algorithms, e.g., the Bareiss algorithm [Bar69], is related to Schur's algorithm for finding the continued fraction representation of a holomorphic function in the unit disk [Sch17]. This class can be generalized to cover unsymmetric matrices and other low displacement rank matrices [KS95a]. In Sections 4.4–4.6 we consider the numerical stability of some of these algorithms. The GKO–Cauchy and GKO–Toeplitz algorithms are discussed in Sec. 4.4. The Schur–Bareiss algorithm for positive-definite matrices is considered in Sec. 4.5.1, and generalized Schur algorithms are mentioned in Sec. 4.5.2. In Sec. 4.6 we consider fast orthogonal factorization algorithms and the fast solution of structured least-squares problems. An embedding approach which leads to a stable algorithm for structured linear systems is mentioned in Sec. 4.6.3.

Algorithms for Vandermonde and many other classes of structured matrices are not considered here—we refer to Ch. 1. Also, we have omitted any discussion of fast "look-ahead" algorithms [CH92a], [CH92b], [FZ93a], [FZ93b], [Gut93], [GH93a], [GH93b], [HG93], [Swe93] because, although such algorithms often succeed in practice, in the worst case they require $O(n^3)$ operations.

Much work has been done on iterative methods for Toeplitz and related systems. Numerical stability is not a major problem with iterative methods, but the speed of convergence depends on a good choice of preconditioner. Iterative methods are considered in Chs. 5 and 7, so we do not consider them in detail here. However, it is worth noting that iterative refinement [JW77], [Wil65] can be used to improve the accuracy of solutions obtained by direct methods (see, e.g., Sec. 2.12.2).

4.1.2 Notation

In the following, R denotes an upper triangular or structured matrix, T is a Toeplitz or Toeplitz-like matrix, P is a permutation matrix, L is lower triangular, U is upper triangular, and Q is orthogonal. If A is a matrix with elements a_{jk}, then $|A|$ denotes the matrix with elements $|a_{jk}|$. In error bounds, ε is the machine precision [GV96], and $O_n(\varepsilon)$ means $O(\varepsilon f(n))$, where $f(n)$ is a polynomial in n. We usually do not try to specify the polynomial $f(n)$ precisely because it depends on the norm used to measure the error and on many unimportant details of the implementation. Also, the $\hat{\cdot}$ notation denotes computed quantities.

4.2 STABILITY AND WEAK STABILITY

In this section we give definitions of stability and weak stability of algorithms for solving linear systems. Consider algorithms for solving a nonsingular, $n \times n$ linear system $Ax = b$. To avoid trivial exceptional cases we always assume that $b \neq 0$.

The *condition number* $\kappa = \kappa(A)$ is defined to be the ratio σ_1/σ_n of the largest and smallest singular values of the matrix A (see [GV96] for a discussion of condition number and singular values). If κ is close to 1 we say that A is *well-conditioned*, and if κ is large we say that A is *ill-conditioned*. The meaning of "large" is a little flexible, but $\kappa > 1/\varepsilon$ is certainly large and, depending on the circumstances, we may regard $\kappa > 1/\sqrt{\varepsilon}$ as large.

There are many definitions of numerical stability in the literature, for example, [Bjo87], [Bjo91], [BBHS95], [BS91], [Bun85], [Cyb80], [GV96], [JW77], [MW80], [Pai73], [Ste73]. Definitions 4.2.1 and 4.2.2 are taken from Bunch [Bun87].

Definition 4.2.1 (Stable Algorithm). *An algorithm for solving linear equations is* stable *for a class of matrices $\mathcal{A}$ if for each A in $\mathcal{A}$ and for each b the computed solution $\widehat{x}$ to $Ax = b$ satisfies $\tilde{A}\widehat{x} = \tilde{b}$, where $\tilde{A}$ is close to A and $\tilde{b}$ is close to b.*

$\diamondsuit$

Definition 4.2.1 says that, for stability, the *computed* solution has to be the *exact* solution of a problem that is close to the original problem. This is the classical *backward stability* of Wilkinson [Wil61], [Wil63], [Wil65]. We interpret "close" to mean close in the relative sense in some norm, i.e.,

$$\|\tilde{A} - A\|/\|A\| = O_n(\varepsilon), \quad \|\tilde{b} - b\|/\|b\| = O_n(\varepsilon).$$

It is well known that the perturbation in A can be absorbed into the perturbation in b, since

$$A\widehat{x} = \tilde{b} + (A - \tilde{A})\widehat{x}\,.$$

Alternatively, the perturbation in b can be absorbed into the perturbation in A, since

$$(\tilde{A} - ||\widehat{x}||_2^{-2}(\tilde{b} - b)\widehat{x}^T)\widehat{x} = b\,.$$

Thus, it is not necessary to permit perturbations of both A and b in Definition 4.2.1.

Note that the matrix $\tilde{A}$ is not required to be in the class $\mathcal{A}$. For example, $\mathcal{A}$ might be the class of nonsingular Toeplitz matrices, but $\tilde{A}$ is not required to be a Toeplitz matrix. If we require $\tilde{A} \in \mathcal{A}$ we get what Bunch [Bun87] calls *strong stability.* For a discussion of the difference between stability and strong stability for Toeplitz algorithms, see [GK93], [GKX94], [HH92], [Var92].

Stability does not imply that the computed solution $\widehat{x}$ is close to the exact solution x, unless the problem is well-conditioned. Provided $\kappa\varepsilon$ is sufficiently small, stability implies that

$$\|\widehat{x} - x\|/\|x\| = O_n(\kappa\varepsilon). \tag{4.2.1}$$

For more precise results, see Bunch [Bun87] and Wilkinson [Wil61].

4.2.1 Gaussian Elimination with Pivoting

To provide a basis for comparison when we quote error bounds for fast methods, and to give an example of a method for which the error bounds are necessarily more pessimistic than what is usually observed, it is worthwhile to consider briefly the classical method of Gaussian elimination (see also Sec. 1.6.2). Wilkinson [Wil61] shows that

$$\|\tilde{A} - A\|/\|A\| = O_n(g\varepsilon),$$

where $g = g(n)$ is the "growth factor." g depends on whether partial or complete pivoting is used. In practice g is usually moderate, even for partial pivoting. However, a well-known example shows that $g(n) = 2^{n-1}$ is possible for partial pivoting, and it has been shown that examples where $g(n)$ grows exponentially with n may occasionally arise in applications, e.g., for linear systems arising from boundary value problems [HH89].

Even for complete pivoting, it has not been *proved* that $g(n)$ is bounded by a polynomial in n. Wilkinson [Wil61] showed that $g(n) \leq n^{(\log n)/4+O(1)}$, and Gould [Gou91]

showed that $g(n) > n$ is possible for $n > 12$; there is still a large gap between these results. Thus, to ensure that Gaussian elimination satisfies Definition 4.2.1, we must restrict $\mathcal{A}$ to the class of matrices for which g is $O_n(1)$. In practice this is not a serious problem, except in certain safety-critical real-time applications, because g can be computed easily and cheaply as the computation proceeds. In the unlikely event that g is large, the result can be checked by an independent computation using a more stable (but slower) method.

4.2.2 Weak Stability

Although stability is desirable, it is more than we can prove for many useful algorithms. Thus, following Bunch [Bun87], we define the (weaker, but still useful) property of *weak stability.*

Definition 4.2.2 (Weak Stability). *An algorithm for solving linear equations is* weakly stable *for a class of matrices $\mathcal{A}$ if for each well-conditioned A in $\mathcal{A}$ and for each b the computed solution $\widehat{x}$ to $Ax = b$ is such that $\|\widehat{x} - x\|/\|x\|$ is small.*

◇

In Def. 4.2.2, we take "small" to mean $O_n(\varepsilon)$ and "well-conditioned" to mean that $\kappa(A)$ is $O_n(1)$, i.e., is bounded by a polynomial in n. From (4.2.1), stability implies weak stability. It may happen that we cannot prove the inequality (4.2.1), but we can prove the weaker inequality

$$||\widehat{x} - x||/||x|| = O_n(\kappa^2 \varepsilon).$$

Clearly, in such a case, the method is weakly stable.

Define the *residual vector* r by

$$r \triangleq A\widehat{x} - b\,. \tag{4.2.2}$$

We refer to $r/||b||$ as the *normalized residual.* It is easy to compute and gives an indication of how well the numerical method has performed—we would be unhappy if the the size of the normalized residual was large, and in general the smaller it is the better. It is important to realize that a small normalized residual does not necessarily mean the that computed solution $\widehat{x}$ is accurate. If the condition number $\kappa(A)$ is large, then $\widehat{x}$ might not agree with the "correct" solution x to any significant figures, although the normalized residual is very small. The residual and the solution error satisfy the following well-known inequalities [Wil63]:

$$\frac{1}{\kappa}\frac{\|r\|}{\|b\|} \leq \frac{\|\widehat{x} - x\|}{\|x\|} \leq \kappa \frac{\|r\|}{\|b\|}. \tag{4.2.3}$$

Thus, for *well-conditioned* A, $\|\widehat{x} - x\|/\|x\|$ is small if and only if $\|r\|/\|b\|$ is small. This observation leads to an alternative definition of weak stability.

Definition 4.2.3 (Equivalent Condition). *An algorithm for solving linear equations is* weakly stable *for a class of matrices $\mathcal{A}$ if for each well-conditioned A in $\mathcal{A}$ and for each b the computed solution $\widehat{x}$ to $Ax = b$ is such that $\|A\widehat{x} - b\|/\|b\|$ is small.*

◇

If we can prove that

$$||A\widehat{x} - b||/||b|| = O_n(\varepsilon),$$

then the method is stable; if we can prove that

$$||A\widehat{x} - b||/||b|| = O_n(\kappa\varepsilon),$$

then the method is (at least) weakly stable.

4.2.3 Example: Orthogonal Factorization

To illustrate the concepts of stability and weak stability, consider computation of the Cholesky factor R of A^TA,[7] where A is an $m \times n$ matrix of full rank n. For simplicity we assume that $||A^TA||$ is of order unity. A good $O(mn^2)$ algorithm is to compute the QR factorization

$$A = QR$$

of A using Householder or Givens transformations [GV96] (see also App. B). It can be shown [Wil63] that the computed matrices $\widehat{Q}$, $\widehat{R}$ satisfy

$$\tilde{A} = \tilde{Q}\widehat{R}, \tag{4.2.4}$$

where $\tilde{Q}^T\tilde{Q} = I$, $\widehat{Q}$ is close to $\tilde{Q}$, and $\tilde{A}$ is close to A. Thus, the algorithm is stable in the sense of backward error analysis. Note that $\|A^TA - \widehat{R}^T\widehat{R}\|$ is small, but $\|\widehat{Q} - Q\|$ and $\|\widehat{R} - R\|/\|R\|$ are not necessarily small. Bounds on $\|\widehat{Q} - Q\|$ and $\|\widehat{R} - R\|/\|R\|$ depend on $\kappa(A)$ and are discussed in [Gol65], [Ste77], [Wil65].

A different algorithm is to compute (the upper triangular part of) A^TA and then compute the Cholesky factorization of A^TA by the usual (stable) algorithm. The computed result $\widehat{R}$ is such that $\widehat{R}^T\widehat{R}$ is close to A^TA. However, this does not imply the existence of $\tilde{A}$ and $\tilde{Q}$ such that (4.2.4) holds (with $\tilde{A}$ close to A and some $\tilde{Q}$ with $\tilde{Q}^T\tilde{Q} = I$) unless A is well-conditioned [Ste79]. By analogy with Definition 4.2.3 above, we may say that Cholesky factorization of A^TA gives a *weakly stable* algorithm for computing R, because the "residual" $A^TA - \widehat{R}^T\widehat{R}$ is small.

4.3 CLASSES OF STRUCTURED MATRICES

We consider structured matrices R that satisfy displacement equations of the form[8]

$$\nabla_{\{F,A\}}(R) = FR - RA^T = GB^T, \tag{4.3.1}$$

where F and A have some simple structure (usually lower triangular or banded, with three or fewer full diagonals), G and B are $n \times \alpha$, and α is some fixed integer. The pair of matrices (G, B) is called an $\{F, A\}$-*generator* of R.

The number α is called the *displacement rank* of R with respect to the displacement operation (4.3.1). We are interested in cases where α is small (say, at most 4). For a discussion of the history and some variants of (4.3.1), see Ch. 1 of this book and [KS95a].

[7] In this subsection, the symbol R denotes an upper triangular matrix.

[8] We now use the letter R to denote a structured matrix and the letter A to denote a displacement operator.

4.3.1 Cauchy and Cauchy-Like Matrices

Particular choices of F and A lead to definitions of basic classes of matrices (see also Sec. 1.3). Thus, for a Cauchy matrix, $\alpha = 1$,

$$R = C(\mathbf{t}, \mathbf{s}) \triangleq \left[\frac{1}{t_j - s_k}\right]_{jk},$$

we have

$$F = D_t \triangleq \operatorname{diag}(t_1, t_2, \ldots, t_n),$$

$$A = D_s \triangleq \operatorname{diag}(s_1, s_2, \ldots, s_n),$$

and

$$G^T = B^T = [1, 1, \ldots, 1].$$

As a natural generalization, we can take G and B to be any $n \times \alpha$ rank-α matrices, with F and A as above. Then a matrix R satisfying (4.3.1) is said to be a *Cauchy-like* matrix.

4.3.2 Toeplitz Matrices

For a Toeplitz matrix $R = T = [t_{jk}] = [a_{j-k}]$, we can take $\alpha = 2$,

$$F = Z_1 \triangleq \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & & & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}, \quad A^T = Z_{-1} \triangleq \begin{bmatrix} 0 & 0 & \cdots & 0 & -1 \\ 1 & 0 & & & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix},$$

$$G^T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_0 & a_{1-n} + a_1 & \cdots & a_{-1} + a_{n-1} \end{bmatrix}, \tag{4.3.2}$$

and

$$B^T = \begin{bmatrix} a_{n-1} - a_{-1} & \cdots & a_1 - a_{1-n} & a_0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}. \tag{4.3.3}$$

We can generalize to *Toeplitz-like* matrices by taking G and B to be general $n \times \alpha$ rank-α matrices, $\alpha \geq 2$. We can also choose other matrices $\{F, A\}$ for the Toeplitz case (see Ch. 1).

4.4 STRUCTURED GAUSSIAN ELIMINATION

Let an input matrix, R_1, have the partitioning

$$R_1 = \begin{bmatrix} d_1 & \mathbf{w}_1^T \\ \mathbf{y}_1 & \tilde{R}_1 \end{bmatrix}.$$

The first step of normal Gaussian elimination is to premultiply R_1 by (see also Sec. 1.6.2 for a related discussion)

$$\begin{bmatrix} 1 & 0 \\ -\mathbf{y}_1/d_1 & I \end{bmatrix},$$

which reduces it to

$$\begin{bmatrix} d_1 & \mathbf{w}_1^T \\ 0 & R_2 \end{bmatrix},$$

where

$$R_2 = \tilde{R}_1 - \mathbf{y}_1 \mathbf{w}_1^T / d_1$$

is the *Schur complement* of d_1 in R_1. At this stage, R_1 has the factorization

$$R_1 = \begin{bmatrix} 1 & 0 \\ \mathbf{y}_1/d_1 & I \end{bmatrix} \begin{bmatrix} d_1 & \mathbf{w}_1^T \\ 0 & R_2 \end{bmatrix}.$$

One can proceed recursively with the Schur complement R_2, eventually obtaining a factorization $R_1 = LU$.

As discussed in Sec. 1.5, the key to *structured* Gaussian elimination is the fact that the displacement structure is preserved under Schur complementation and that the generators for the Schur complement of R_{k+1} can be computed from the generators of R_k in $O(n)$ operations (see also Sec. 1.12).

Row or column interchanges destroy the structure of matrices such as Toeplitz matrices. However, if F is diagonal (which is the case for Cauchy- and Vandermonde-type matrices), then *the structure is preserved under row permutations* (recall the discussion in Sec. 1.13). This observation leads to the *GKO–Cauchy* algorithm of [GKO95] for fast factorization of Cauchy-like matrices with partial pivoting (see Sec. 1.13.1). The idea of using pivoting in a hybrid Schur–Levinson algorithm for Cauchy-like systems was introduced by Heinig [Hei95]. There have been several variations on the idea in recent papers [GKO95], [Gu95a], [Ste98].

4.4.1 The GKO–Toeplitz Algorithm

Heinig [Hei95] showed that, if T is a Toeplitz-like matrix with $\{Z_1, Z_{-1}^T\}$-generators (G, B), then

$$R \triangleq \mathcal{F} T D^* \mathcal{F}^*$$

is a Cauchy-like matrix, where

$$\mathcal{F} \triangleq \frac{1}{\sqrt{n}} \left[e^{2\pi \hat{\imath}(j-1)(k-1)/n} \right]_{1 \le j,k \le n}, \quad \hat{\imath} = \sqrt{-1},$$

is the discrete Fourier transform (DFT) matrix,

$$D = \operatorname{diag}\left(1, e^{\pi \hat{\imath}/n}, \ldots, e^{\pi \hat{\imath}(n-1)/n}\right),$$

and the generators of T and R are simply related. In fact, it is easy to verify that

$$\nabla_{\{D_f, D_a\}}(R) = (\mathcal{F}G)(B^T D^* \mathcal{F}^*), \tag{4.4.1}$$

where

$$D_f \triangleq D^2 = \operatorname{diag}\left(1, e^{2\pi \hat{\imath}/n}, \ldots, e^{2\pi \hat{\imath}(n-1)/n}\right)$$

and

$$D_a \triangleq e^{\pi \hat{\imath}/n} D_f = \operatorname{diag}\left(e^{\pi \hat{\imath}/n}, e^{3\pi \hat{\imath}/n}, \ldots, e^{(2n-1)\pi \hat{\imath}/n}\right).$$

Thus, the (D_f, D_a) generator of R is $(\mathcal{F}G, \mathcal{F}DB)$.

The transformation $T \leftrightarrow R$ is perfectly stable because $\mathcal{F}$ and D are unitary. Note that R is (in general) complex even if T is real. This increases the constant factors in the time bounds, because complex arithmetic is required.

Heinig's observation was exploited in [GKO95] (see Sec. 1.13.1): R can be factored as $R = P^T LU$ using the GKO–Cauchy algorithm. Thus, from the factorization

$$T = \mathcal{F}^* P^T LUFD,$$

a linear system involving T can be solved in $O(n^2)$ operations. The full procedure of conversion to Cauchy form, factorization, and solution requires $O(n^2)$ complex operations.

Other structured matrices, such as Hankel, Toeplitz-plus-Hankel, Vandermonde, Chebyshev–Vandermonde, etc., can be converted to Cauchy-like matrices in a similar way (e.g., [GKO95], [KO95]).

4.4.2 Error Analysis

Because GKO–Cauchy and GKO–Toeplitz involve partial pivoting, we might guess that their stability would be similar to that of Gaussian elimination with partial pivoting. Unfortunately, there is a flaw in this reasoning. During GKO–Cauchy the *generators* have to be transformed, and the partial pivoting does not ensure that the transformed generators are small.

Sweet and Brent [SB95] show that significant generator growth can occur if all the elements of GB^T are small compared with those of $|G| \cdot |B^T|$. This cannot happen for ordinary Cauchy matrices because $\alpha = 1$ and the successive generator matrices, G_k and B_k, have only one column and one row, respectively. However, it can happen for higher displacement-rank Cauchy-like matrices, even if the original matrix is well-conditioned.

For example, taking $\alpha = 2$,

$$G = [\mathbf{a}, \mathbf{a} + \mathbf{f}], \qquad B = [\mathbf{a} + \mathbf{e}, -\mathbf{a}],$$

where $||\mathbf{a}||$ is of order unity and $||\mathbf{e}||$ and $||\mathbf{f}||$ are very small, we see that all the elements of $GB^T = \mathbf{a}\mathbf{e}^T - \mathbf{f}\mathbf{a}^T$ are very small compared with those of $|G||B^T|$. Moreover, because $\mathbf{a}$, $\mathbf{e}$, and $\mathbf{f}$ can be arbitrary except for their norms, the original Cauchy-type matrix is in general well-conditioned. The problem is that the *generators* are ill-conditioned, being close to lower-rank matrices.

There are corresponding examples for Toeplitz-type matrices, easily derived using the correspondence between generators of Toeplitz-type and Cauchy-type matrices discussed in Sec. 4.4.1. However, in the strictly Toeplitz case the special form of the matrices B and G in (4.3.2) and (4.3.3) imposes additional constraints, which appear to rule out this kind of example. However, it is still possible to give examples where the normalized solution error grows like κ^2 and the normalized residual grows like κ, where κ is the condition number of the Toeplitz matrix: see Sweet and Brent [SB95, §5.2]. Thus, the GKO–Toeplitz algorithm is (at best) weakly stable.

It is easy to think of modified algorithms that avoid the examples given above and by Sweet and Brent [SB95], but it is difficult to prove that they are stable in all cases. Stability depends on the worst case, which may be rare and hard to find by random sampling.

The problem with the original GKO algorithm is growth in the generators. Gu [Gu95a] suggested exploiting the fact that the generators are not unique. Recall the displacement equation (4.3.1). Clearly we can replace G by GM and B^T by $M^{-1}B^T$,

where M is any invertible $\alpha \times \alpha$ matrix, because this does not change the product GB^T. This holds similarly at later stages of the GKO algorithm. Gu [Gu95a] proposes taking M to orthogonalize the columns of G (that is, at each stage perform an orthogonal factorization of the generators—see also Sec. 2.11). Stewart [Ste98] proposes a (cheaper) LU factorization of the generators. This avoids examples of the type given above. In both cases, clever pivoting schemes give error bounds analogous to those for Gaussian elimination with partial pivoting.

The error bounds obtained by Gu and Stewart involve a factor K^n, where K depends on the ratio of the largest to smallest modulus elements in the Cauchy matrix

$$\left[\frac{1}{t_j - s_k}\right]_{jk}.$$

Although this is unsatisfactory, it is similar to the factor 2^{n-1} in the error bound for Gaussian elimination with partial pivoting. As mentioned in Sec. 4.2.1, the latter factor is extremely pessimistic, which explains why Gaussian elimination with partial pivoting is popular in practice [Hig96]. Perhaps the bounds of Gu and Stewart are similarly pessimistic, although practical experience is not yet extensive enough to be confident of this. Stewart [Ste98] gives some interesting numerical results suggesting that his scheme works well, but more numerical experience is necessary before a definite conclusion can be reached.

4.4.3 A General Strategy

It often happens that there is a choice of

1. a fast algorithm that usually gives an accurate result but occasionally fails (or at least cannot be proved to succeed every time), or
2. an algorithm that is guaranteed to be stable but is slow.

In such cases, a good strategy may be to use the fast algorithm but then check the normalized residual $||A\widehat{x} - b||/||b||$, where $\widehat{x}$ is the computed solution of the system $Ax = b$. If the normalized residual is sufficiently small (say at most 1000ε) we can accept $\widehat{x}$ as a reasonably good solution. In the rare cases that the normalized residual is not sufficiently small we can can use the slow but stable algorithm. (Alternatively, if the residual is not too large, one or two iterations of iterative refinement may be sufficient and faster [JW77], [Wil65]—see also Sec. 2.12.2.)

An example of this general strategy is the solution of a Toeplitz system by Gu or Stewart's modification of the GKO algorithm. We can use the $O(n^2)$ algorithm, check the residual, and resort to iterative refinement or a slow but stable algorithm in the (rare) cases that it is necessary. Computing the residual takes only $O(n \log n)$ arithmetic operations.

We now turn our attention to another class of methods.

4.5 POSITIVE-DEFINITE STRUCTURED MATRICES

An important class of algorithms, typified by the algorithm of Bareiss [Bar69], finds an LU factorization of a Toeplitz matrix T and (in the symmetric case) is related to the classical algorithm of Schur [Bur75], [Goh86], [Sch17] (see also Ch. 1).

It is interesting to consider the numerical properties of these algorithms and compare them with the numerical properties of the Levinson–Durbin algorithm, which essentially finds an LU factorization of T^{-1}.

4.5.1 The Bareiss Algorithm for Positive-Definite Matrices

Bojanczyk, Brent, de Hoog, and Sweet (BBHS) have shown in [BBHS95], [Swe82] that the numerical properties of the Bareiss algorithm are similar to those of Gaussian elimination (*without* pivoting). Thus, the algorithm is stable for positive-definite symmetric Toeplitz matrices. In fact, the results of [BBHS95] establish stability for the larger class of quasi-Toeplitz positive-definite matrices. (For a definition of this class, see Ch. 1.)

The result of [BBHS95, Sec. 5] is that the computed upper triangular factor $\widehat{U}$ of the positive-definite matrix T satisfies the backward error bound

$$||T - \widehat{U}^T\widehat{U}||_2/t_0 = O(n^2\varepsilon)\,, \tag{4.5.1}$$

where t_0 is a normalizing factor (a diagonal element of T), provided the Cholesky downdating is done using the "mixed downdating" scheme [BBDH87]. If "hyperbolic downdating" is used, then the bound (4.5.1) increases to $O(n^3\varepsilon)$, although numerical experiments reported in [BBHS95, Sec. 7] did not demonstrate much difference between the two forms of downdating. (Different forms of downdating are equivalent to different implementations of hyperbolic rotations.) The same numerical experiments showed that Cholesky factorization usually gave a slightly more accurate solution, but a slightly larger residual, than the fast algorithm using either form of downdating.

The Levinson–Durbin algorithm can be shown to be weakly stable for bounded n, and numerical results by [Var93], [BBHS95], and others suggest this is all we can expect. Thus, the Bareiss algorithm is (generally) better numerically than the Levinson–Durbin algorithm. For example, the numerical results reported in [BBHS95, Sec. 7] for three ill-conditioned positive-definite Toeplitz matrices (the Prolate matrix [Var93] and two matrices whose reflection coefficients have alternating signs) indicate that the Levinson–Durbin algorithm typically gives solution errors more than twice as large as the other algorithms (Bareiss with two forms of downdating and Cholesky), and residual errors 4×10^3 to 5×10^5 as large. The condition numbers of the test matrices ranged from 3×10^{14} to 8×10^{15}, and the machine precision was $\varepsilon = 2^{-53} \simeq 10^{-16}$; for better conditioned matrices the differences between the methods are generally less apparent.

Cybenko [Cyb80] showed that if the so-called reflection coefficients (see Sec. 1.2) are all positive, then the Levinson–Durbin algorithm for solving the Yule–Walker equations (a positive-definite Toeplitz system with a special right-hand side) is stable. Unfortunately, Cybenko's result usually is not applicable, because most positive-definite Toeplitz matrices (e.g., the examples just quoted) do not satisfy the restrictive condition on the reflection coefficients.

4.5.2 Generalized Schur Algorithms

The Schur algorithm can be generalized to factor a large variety of structured matrices [KC94], [KS95a] (see Ch. 1). For example, suitably generalized Schur algorithms apply to block Toeplitz matrices, Toeplitz block matrices, and matrices of the form T^TT, where T is rectangular Toeplitz.

It is natural to ask to what extent the stability results of [BBHS95] can be generalized. This has been considered in recent papers by Stewart and Van Dooren [SD97b] and by Chandrasekaran and Sayed [CS96] (see also Chs. 2 and 3 of this book).

Stewart and Van Dooren [SD97b] generalized the results of BBHS to matrices with similar structure but displacement rank larger than 2. They showed that for higher displacement ranks one has to be careful to implement the hyperbolic rotations properly. If they are implemented correctly (e.g., as recommended in [BBDH87]), then a backward

error bound of the same form as (4.5.1) holds. In contrast, as mentioned in Sec. 4.5.1, one does not have to be so careful when the displacement rank is 2 because a result of the same form as (4.5.1), albeit with an extra power of n, holds for other implementations of hyperbolic rotations. (Of course, the error bound will certainly fail for a *sufficiently bad* implementation of hyperbolic rotations.)

Chandrasekaran and Sayed [CS96] studied a significantly more general Schur-like algorithm. (Their results are discussed in Chs. 2 and 3 of this book.) They dropped the assumption that the matrices F, A of Sec. 4.3.2 are shift matrices Z, although the corresponding matrices still have to satisfy certain restrictions. This extra generality causes significant complications in the error analysis and appears to make the Schur algorithm even more sensitive to the method of implementing hyperbolic rotations. Provided these hyperbolic rotations and certain "Blaschke factors" (see Sec. 1.6 for a description of these factors) are implemented correctly, a backward stability result similar to that of BBHS can be established. The interested reader is referred to [CS96] (and also Ch. 2) for details.

The overall conclusion is that the generalized Schur algorithm is stable for positive-definite symmetric (or Hermitian) matrices, *provided* the hyperbolic rotations and the Blaschke factors (if any) in the algorithm are implemented correctly.

We now drop the assumption of positive definiteness, and even (temporarily) the assumption that the matrix T is square, and we consider fast algorithms for orthogonal factorization.

4.6 FAST ORTHOGONAL FACTORIZATION

In an attempt to achieve stability without pivoting, and to solve $m \times n$ least squares problems ($m \geq n$), it is natural to consider algorithms for computing an orthogonal factorization

$$T = QU \tag{4.6.1}$$

of an $m \times n$ Toeplitz matrix T. We assume that T has full rank n. For simplicity, in the time bounds we assume $m = O(n)$ to avoid functions of both m and n.

The first $O(n^2)$ (more precisely, $O(mn)$) algorithm for computing the factorization (4.6.1) was introduced by Sweet [Swe82]. Unfortunately, Sweet's algorithm is unstable—see [LQ87].

Other $O(n^2)$ algorithms for computing the matrices Q and U or U^{-1} were given by Bojanczyk, Brent, and de Hoog (BBH) [BBH86], Chun, Kailath, and Lev-Ari [CKL87], Cybenko [Cyb87], and Qiao [Qia88], but none of them has been shown to be stable, and in several cases examples show that they are unstable.

It may be surprising that fast algorithms for computing an orthogonal factorization (4.6.1) are unstable. The classical $O(n^3)$ algorithms are stable because they form Q as a product of elementary orthogonal matrices (usually Givens or Householder matrices [Gol65], [GV96], [Wil65]). Unlike the classical algorithms, the $O(n^2)$ algorithms do not form Q in a numerically stable manner as a product of matrices that are (close to) orthogonal. This observation explains both their speed and their instability!

For example, the algorithms of [BBH86] and [CKL87] depend on Cholesky downdating, and numerical experiments show that they do not give a Q that is close to orthogonal. This is not too surprising, because Cholesky downdating is known to be a sensitive numerical problem [BBDH87], [Ste79]. Perhaps it is more surprising that the authors of [BBHS95] were able to use an error analysis of a form of downdating in their analysis of the Bareiss algorithm (but only in the positive-definite case)—see Sec. 4.5.1.

4.6.1 Use of the Seminormal Equations

It can be shown that, provided the Cholesky downdates are implemented in a certain way (analogous to the condition for the stability of the Schur algorithm for $\alpha > 2$, discussed in Sec. 4.5.2), the BBH algorithm computes U in a weakly stable manner [BBH95]. In fact, the computed upper triangular matrix $\widehat{U}$ is about as good as can be obtained by performing a Cholesky factorization of $T^T T$, so

$$\|T^T T - \widehat{U}^T \widehat{U}\| / \|T^T T\| = O_m(\varepsilon) .$$

Thus, by solving

$$\widehat{U}^T \widehat{U} x = T^T b$$

(the so-called seminormal equations) we have a *weakly stable* algorithm for the solution of general full-rank Toeplitz least-squares problems. In the case $m = n$, this gives a weakly stable algorithm for the solution of Toeplitz linear systems $Tx = b$ in $O(n^2)$ operations. The solution can be improved by iterative refinement if desired [GW66]. Weak stability is achieved because the computation of Q is avoided. The disadvantage of this method is that, by implicitly forming $T^T T$, the condition of the problem is effectively squared. If the condition number $\kappa = \kappa(T)$ is in the range

$$\frac{1}{\sqrt{\varepsilon}} \leq \kappa \leq \frac{1}{\varepsilon},$$

then it usually will be impossible to get any significant figures in the result (iterative refinement may fail to converge) without reverting to a slow but stable orthogonal factorization algorithm. One remedy is to use double-precision arithmetic, i.e., replace ε by ε^2, but this may be difficult if ε already corresponds to the maximum precision implemented in hardware.

Another way of computing the upper triangular matrix U, but not the orthogonal matrix Q, in (4.6.1) is to apply a generalized Schur algorithm to $T^T T$. This method also squares the condition number.

4.6.2 Computing Q Stably

It seems difficult to give a satisfactory $O(n^2)$ algorithm for the computation of Q in the factorization (4.6.1). The algorithm suggested in [BBH86] is unsatisfactory, as are all other fast algorithms known to us. In Sec. 4.6.1 we sidestepped this problem by avoiding the computation of Q, but in some applications Q is required. We leave the existence of a fast, stable algorithm for the computation of Q as an open question.

4.6.3 Solution of Indefinite or Unsymmetric Structured Systems

Using a modification of the embedding approach pioneered by Chun and Kailath [Chu89], [KC94] and Chandrasekaran and Sayed [CS98] gives a stable algorithm to compute a factorization of the form (see Ch. 3 of this book)

$$T = \Delta(\Delta^{-1} Q) U \tag{4.6.2}$$

in terms of three matrices $\{\Delta, Q, U\}$, where Δ is a lower triangular matrix that compensates for the fact that Q is not numerically orthogonal. (Of course, the factorization (4.6.2) is not unique.) The motivation of [CS98] was to give a backward-stable

algorithm for general square Toeplitz or quasi-Toeplitz systems (symmetric but not positive definite, or possibly unsymmetric). For details of the stability analysis, we refer to [CS98] and Ch. 3.

The algorithm of [CS98] can be used to solve linear equations but not least-squares problems (because T has to be square). Because the algorithm involves embedding the $n \times n$ matrix T in a $2n \times 2n$ matrix

$$\begin{bmatrix} T^T T & T^T \\ T & 0 \end{bmatrix},$$

the constant factors in the operation count are moderately large: $59n^2 + O(n \log n)$, which should be compared to $8n^2 + O(n \log n)$ for BBH and the seminormal equations (a weakly stable method, as discussed in Sec. 4.6.1). These operation counts apply for $m = n$: see [BBH95] for operation counts of various algorithms when $m \geq n$.

4.7 CONCLUDING REMARKS

Although this survey has barely scratched the surface, we hope the reader who has come this far is convinced that questions of numerical stability are amongst the most interesting, difficult, and useful questions that we can ask about fast algorithms for structured linear systems. It is not too hard to invent a new fast algorithm, but to find a new stable algorithm is more difficult, and to *prove* its stability or weak stability is a real challenge!

Acknowledgments

A preliminary version of this review appeared in [Bre97]. Thanks to Greg Ammar, Adam Bojanczyk, James Bunch, Shiv Chandrasekaran, George Cybenko, Paul Van Dooren, Lars Eldén, Roland Freund, Andreas Griewank, Ming Gu, Martin Gutknecht, Georg Heinig, Frank de Hoog, Franklin Luk, Vadim Olshevsky, Haesun Park, A. H. Sayed, Michael Stewart, Douglas Sweet, and James Varah for their assistance, and especially to A. H. Sayed for his detailed comments on a draft of this chapter.