Chapter 1

DISPLACEMENT STRUCTURE AND ARRAY ALGORITHMS

Thomas Kailath

1.1 INTRODUCTION

Many problems in engineering and applied mathematics ultimately require the solution of $n \times n$ linear systems of equations. For small-size problems, there is often not much else to do except to use one of the already standard methods of solution such as Gaussian elimination. However, in many applications, n can be very large ($n \sim 1000, n \sim 1,000,000$) and, moreover, the linear equations may have to be solved over and over again, with different problem or model parameters, until a satisfactory solution to the original physical problem is obtained. In such cases, the $O(n^3)$ burden, i.e., the number of flops required to solve an $n \times n$ linear system of equations, can become prohibitively large. This is one reason why one seeks in various classes of applications to identify special or characteristic structures that may be assumed in order to reduce the computational burden. Of course, there are several different kinds of structure.

A special form of structure, which already has a rich literature, is sparsity; i.e., the coefficient matrices have only a few nonzero entries. We shall not consider this already well studied kind of structure here. Our focus will be on problems, as generally encountered in communications, control, optimization, and signal processing, where the matrices are not sparse but can be very large. In such problems one seeks further assumptions that impose particular patterns among the matrix entries. Among such assumptions (and we emphasize that they are always assumptions) are properties such as time-invariance, homogeneity, stationarity, and rationality, which lead to familiar matrix structures, such as Toeplitz, Hankel, Vandermonde, Cauchy, Pick, etc. Several fast algorithms have been devised over the years to exploit these special structures. The numerical (accuracy and stability) properties of several of these algorithms also have been studied, although, as we shall see from the chapters in this volume, the subject is by no means closed even for such familiar objects as Toeplitz and Vandermonde matrices.

In this book, we seek to broaden the above universe of discourse by noting that even more common than the *explicit* matrix structures, noted above, are matrices in which the structure is *implicit*. For example, in least-squares problems one often encounters products of Toeplitz matrices; these products generally are not Toeplitz, but on the other hand they are not "unstructured." Similarly, in probabilistic calculations the matrix of interest often is not a Toeplitz covariance matrix, but rather its inverse, which is rarely

Toeplitz itself, but of course is not unstructured: its inverse is Toeplitz. It is well known that $O(n^2)$ flops suffice to solve linear systems with an $n \times n$ Toeplitz coefficient matrix; a question is whether we will need $O(n^3)$ flops to invert a non-Toeplitz coefficient matrix whose inverse is known to be Toeplitz. When pressed, one's response clearly must be that it is conceivable that $O(n^2)$ flops will suffice, and we shall show that this is in fact true.

Such problems, and several others that we shall encounter in later chapters, suggest the need for a *quantitative* way of defining and identifying structure in (dense) matrices. Over the years we have found that an elegant and useful way is the concept of *displacement structure*. This has been useful for a host of problems apparently far removed from the solution of linear equations, such as the study of constrained and unconstrained rational interpolation, maximum entropy extension, signal detection, system identification, digital filter design, nonlinear Riccati differential equations, inverse scattering, certain Fredholm and Wiener–Hopf integral equations, etc. However, in this book we shall focus attention largely on displacement structure in matrix computations. For more general earlier reviews, we may refer to [KVM78], [Kai86], [Kai91], [HR84], [KS95a].

1.2 TOEPLITZ MATRICES

The concept of displacement structure is perhaps best introduced by considering the much-studied special case of a Hermitian Toeplitz matrix,

$$T = \begin{bmatrix} c_0 & c_{-1} & c_{-2} & \cdots & c_{-n+1} \\ c_1 & c_0 & c_{-1} & \cdots & c_{-n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}, \quad c_k = c_{-k}^*. \tag{1.2.1}$$

The matrix T has constant entries along its diagonals and, hence, it depends only on n parameters rather than n^2. As stated above, it is therefore not surprising that many matrix problems involving T, such as triangular factorization, orthogonalization, and inversion, have solution complexity $O(n^2)$ rather than $O(n^3)$ operations. The issue is the complexity of such problems for inverses, products, and related combinations of Toeplitz matrices such as $T^{-1}, T_1T_2, T_1 - T_2T_3^{-1}T_4, (T_1T_2)^{-1}T_3 \ldots$. As mentioned earlier, although these are not Toeplitz, they are certainly structured and the complexity of inversion and factorization may be expected to be not much different from that for a pure Toeplitz matrix, T. It turns out that the appropriate common property of all these matrices is not their "Toeplitzness," but the fact that they all have (low) *displacement rank* in a sense first defined in [KKM79a], [KKM79b] and later much studied and generalized. When the displacement rank is r, $r \leq n$, the solution complexity of the above problems turns out to be $O(rn^2)$. Now for some formal definitions.

The displacement of a Hermitian matrix $R = [r_{ij}]_{i,j=0}^{n-1} \in \mathbb{C}^{n \times n}$ was originally[1] defined in [KKM79a], [KKM79b] as

$$\nabla_Z R \triangleq R - ZRZ^*, \tag{1.2.2}$$

[1]Other definitions will be introduced later. We may note that the concept was first identified in studying integral equations (see, e.g., [KLM78]).

where $*$ denotes Hermitian conjugation (complex conjugation for scalars) and Z is the $n \times n$ lower shift matrix with ones on the first subdiagonal and zeros elsewhere,

$$Z \triangleq \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ & 1 & 0 & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix}. \tag{1.2.3}$$

The product ZRZ^* then corresponds to shifting R downward along the main diagonal by one position, explaining the name *displacement* for $\nabla_Z R$. The situation is depicted in Fig. 1.1.

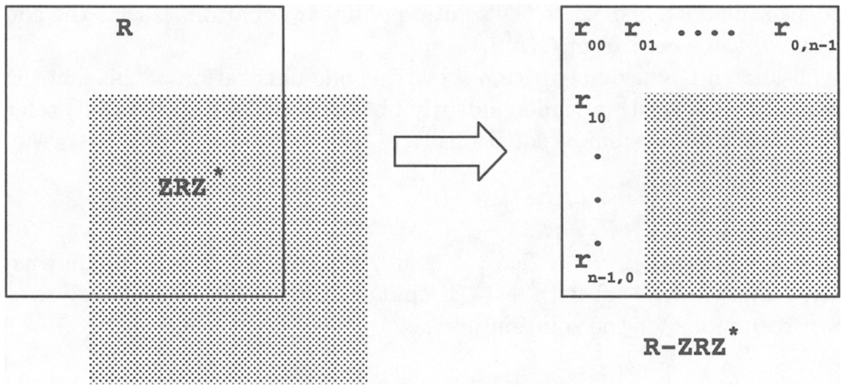

Figure 1.1. $\nabla_Z R$ is obtained by shifting R downward along the diagonal.

If $\nabla_Z R$ has (low) rank, say, r, *independent* of n, then R is said to be *structured* with respect to the displacement ∇_Z defined by (1.2.2), and r is called the *displacement rank* of R. The definition can be extended to non-Hermitian matrices, and this will be briefly described later. Here we may note that in the Hermitian case, $\nabla_Z R$ is Hermitian and therefore has further structure: its eigenvalues are real and so we can define the *displacement inertia* of R as the pair $\{p, q\}$, where p (respectively, q) is the number of strictly positive (respectively, negative) eigenvalues of $\nabla_Z R$. Of course, the displacement rank is $r = p + q$. Therefore, we can write

$$\nabla_Z R = R - ZRZ^* = GJG^*, \tag{1.2.4}$$

where $J = J^* = (I_p \oplus -I_q)$ is a signature matrix and $G \in \mathbb{C}^{n \times r}$. The pair $\{G, J\}$ is called a ∇_Z-*generator* of R. This representation is clearly not unique; for example, $\{G\Theta, J\}$ is also a generator for any J-unitary matrix Θ (i.e., for any Θ such that $\Theta J \Theta^* = J$). This is because

$$G \underbrace{\Theta J \Theta^*}_{J} G^* = GJG^*.$$

Nonminimal generators (where G has more than r columns) are sometimes useful, although we shall not consider them here.

Returning to the Toeplitz matrix (1.2.1), it is easy to see that T has displacement rank 2, except when all c_i, $i \neq 0$, are zero, a case we shall exclude. Assuming

$c_0 = 1$, a generator for T is $\{x_0, y_0, (1 \oplus -1)\}$, where $x_0 = \text{col}\{1, c_1, \ldots, c_{n-1}\}$ and $y_0 = \text{col}\{0, c_1, \ldots, c_{n-1}\}$ (the notation $\text{col}\{\cdot\}$ denotes a column vector with the specified entries):

$$T - ZTZ^* = \begin{bmatrix} 1 & 0 \\ c_1 & c_1 \\ \vdots & \vdots \\ c_{n-1} & c_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ c_1 & c_1 \\ \vdots & \vdots \\ c_{n-1} & c_{n-1} \end{bmatrix}^* . \tag{1.2.5}$$

It will be shown later that if we define $T^{\#} \triangleq \tilde{I}T^{-1}\tilde{I}$, where $\tilde{I}$ denotes the reversed identity with ones on the reversed diagonal and zeros elsewhere, then $T^{\#}$ also has ∇_Z-displacement inertia $\{1, 1\}$. The product T_1T_2 of two Toeplitz matrices, which may not be Hermitian, will be shown to have displacement rank ≤ 4. The significance of displacement rank with respect to the solution of linear equations is that the complexity can be reduced to $O(rn^2)$ from $O(n^3)$.

The well-known Levinson algorithm [Lev47] is one illustration of this fact. The best-known form of this algorithm (independently obtained by Durbin [Dur59]) refers to the so-called Yule–Walker system of equations

$$a_n T_n = \begin{bmatrix} 0 & 0 & \ldots & 0 & \sigma_n^2 \end{bmatrix}, \tag{1.2.6}$$

where $a_n = \begin{bmatrix} a_{n,n} & a_{n,n-1} & \cdots & a_{n,1}, 1 \end{bmatrix}$ and σ_n^2 are the $(n+1)$ unknowns and T_n is a positive-definite $(n+1) \times (n+1)$ Toeplitz matrix. The easily derived and now well-known recursions for the solution are

$$\begin{bmatrix} a_{i+1} \\ a_{i+1}^{\#} \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{i+1} \\ -\gamma_{i+1}^* & 1 \end{bmatrix} \begin{bmatrix} 0 & a_i \\ a_i^{\#} & 0 \end{bmatrix}, \quad a_0 = a_0^{\#} = 1, \tag{1.2.7}$$

where $\gamma_{i+1} = \delta_i / \sigma_i^2$,

$$\delta_i = a_{i,i}c_1 + a_{i,i-1}c_2 + \cdots + a_{i,1}c_i + c_{i+1}, \tag{1.2.8}$$

$$\sigma_{i+1}^2 = \sigma_i^2[1 - |\gamma_{i+1}|^2], \quad \sigma_0^2 = c_0, \tag{1.2.9}$$

and

$$a_i^{\#} = \begin{bmatrix} 1 & a_{i,1}^* & \cdots & a_{i,i}^* \end{bmatrix}.$$

The above recursions are closely related to certain (nonrecursive[2]) formulas given by Szegő [Sze39] and Geronimus [Ger54] for polynomials orthogonal on the unit circle, as discussed in some detail in [Kai91]. It is easy to check that the $\{\gamma_i\}$ are all less than one in magnitude; in signal processing applications, they are often called reflection coefficients (see, e.g., [Kai85], [Kai86]).

While the Levinson–Durbin algorithm is widely used, it has limitations for certain applications. For one thing, it requires the formation of inner products and therefore is not efficiently parallelizable, requiring $O(n \log n)$ rather than $O(n)$ flops, with $O(n)$ processors. Second, while it can be extended to indefinite and even non-Hermitian Toeplitz matrices, it is difficult to extend it to non-Toeplitz matrices having displacement structure. Another problem is numerical. An error analysis of the algorithm in [Cyb80] showed that in the case of positive reflection coefficients $\{\gamma_i\}$, the residual error produced by the Levinson–Durbin procedure is comparable to the error produced

[2]They defined γ_i as $-a_{i+1,i+1}$.

by the numerically well-behaved Cholesky factorization [GV96, p. 191]. Thus in this special case the Levinson–Durbin algorithm is what is called *weakly stable*, in the sense of [Bun85], [Bun87]—see Sec. 4.2 of this book. No stability results seem to be available for the Levinson–Durbin algorithm for Toeplitz matrices with general $\{\gamma_i\}$.

To motivate an alternative (parallelizable and stable) approach to the problem, we first show that the Levinson–Durbin algorithm directly yields a (fast) triangular factorization of T_n^{-1}. To show this, note that stacking the successive solutions of the Yule–Walker equations (1.2.6) in a lower triangular matrix yields the equality

$$\begin{bmatrix} 1 & & & & \\ a_{11} & 1 & & & \\ a_{22} & a_{21} & 1 & & \\ \vdots & & & \ddots & \\ a_{n,n} & a_{n,n-1} & & \dots & 1 \end{bmatrix} T_n \begin{bmatrix} 1 & & & & \\ a_{11} & 1 & & & \\ a_{22} & a_{21} & 1 & & \\ \vdots & & & \ddots & \\ a_{n,n} & a_{n,n-1} & & \dots & 1 \end{bmatrix}^*$$

$$= \begin{bmatrix} \sigma_0^2 & \times & \times & \times & \times \\ & \sigma_1^2 & \times & \times & \times \\ & & \sigma_2^2 & \times & \times \\ & & & \ddots & \times \\ & & & & \sigma_n^2 \end{bmatrix},$$

which, using the Hermitian nature of T, yields the unique triangular factorization of the *inverse* of T_n:

$$T_n^{-1} = \begin{bmatrix} 1 & a_{11}^* & a_{22}^* & \dots & a_{nn}^* \\ & 1 & a_{21}^* & \dots & a_{n,n-1}^* \\ & & & & \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix} D_n^{-1} \begin{bmatrix} 1 & & & & \\ a_{11} & 1 & & & \\ a_{22} & a_{21} & 1 & & \\ \vdots & & & \ddots & \\ a_{n,n} & a_{n,n-1} & & \dots & 1 \end{bmatrix}, \tag{1.2.10}$$

where $D_n = \operatorname{diag}\{\sigma_0^2, \sigma_1^2, \dots, \sigma_n^2\}$.

However, it is a fact, borne out by results in many different problems, that ultimately even for the solution of linear equations the (direct) triangular factorization of T rather than T^{-1} is more fundamental. Such insights can be traced back to the celebrated Wiener–Hopf technique [WH31] but, as perhaps first noted by Von Neumann and by Turing (see, e.g., [Ste73]), direct factorization is the key feature of the fundamental Gaussian elimination method for solving linear equations and was effectively noted as such by Gauss himself (though of course not in matrix notation).

Now a fast direct factorization of T_n cannot be obtained merely by inverting the factorization (1.2.10) of T_n^{-1}, because that will require $O(n^3)$ flops. The first fast algorithm was given by Schur [Sch17], although this fact was realized only much later in [DVK78]. In the meantime, a closely related direct factorization algorithm was derived by Bareiss [Bar69], and this is the designation used in the numerical analysis community. Morf [Mor70], [Mor74], Rissanen [Ris73], and LeRoux–Gueguen [LG77] also made independent rediscoveries.

We shall show in Sec. 1.7.7 that the Schur algorithm can also be applied to solve Toeplitz linear equations, at the cost of about 30% more computations than via the Levinson–Durbin algorithm. However, in return we can compute the reflection coefficients without using inner products, the algorithm has better numerical properties (see Chs. 2–4 and also [BBHS95], [CS96]), and as we shall show below, it can be elegantly and usefully extended by exploiting the concept of displacement structure. These generalizations are helpful in solving the many classes of problems (e.g., interpolation) mentioned earlier (at the end of Sec. 1.1). Therefore, our major focus in this chapter will be on what we have called generalized Schur algorithms.

First, however, let us make a few more observations on displacement structure.

1.3 VERSIONS OF DISPLACEMENT STRUCTURE

There are of course other kinds of displacement structure than those introduced in Sec. 1.2, as already noted in [KKM79a]. For example, it can be checked that

$$\operatorname{rank}\,(Z_{-1}T - TZ) = 2, \tag{1.3.1}$$

where Z_{-1} denotes the circulant matrix with first row $\begin{bmatrix} 0 & \dots & 0 & -1 \end{bmatrix}$. This fact has been used by Heinig [Hei95] and by [GKO95] to obtain alternatives to the Levinson–Durbin and Schur algorithms for solving Toeplitz systems of linear equations, as will be discussed later in this chapter (Sec. 1.13). However, a critical point is that because Z_{-1} is not triangular, these methods apply only to fixed n, and the whole solution has to be repeated if the size is increased even by one. Since many applications in communications, control, and signal processing involve continuing streams of data, recursive triangular factorization is often a critical requirement. It can be shown [LK86] that such factorization requires that triangular matrices be used in the definition of displacement structure, which is what we shall do henceforth.

One of the first extensions of definition (1.2.2) was to consider

$$\nabla_F R \overset{\Delta}{=} R - FRF^*, \tag{1.3.2}$$

where, for reasons mentioned above, F is a lower triangular matrix; see [LK84], [CKL87]. One motivation for such extensions will be seen in Sec. 1.8. Another is that one can include matrices such as Vandermonde, Cauchy, Pick, etc. For example, consider the so-called Pick matrix, which occurs in the study of analytic interpolation problems,

$$P = \left[\frac{u_i u_j^* - v_j v_j^*}{1 - f_i f_j^*}\right]_{i,j=0}^{n-1},$$

where $\{u_i, v_i\}$ are row vectors of dimensions p and q, respectively, and f_i are complex points inside the open unit disc ($|f_i| < 1$). If we let F denote the diagonal matrix $\operatorname{diag}\{f_0, f_1, \dots, f_{n-1}\}$, then it can be verified that P has displacement rank $(p+q)$ with respect to F since

$$P - FPF^* = \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \end{bmatrix} \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix} \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \end{bmatrix}^* .$$

In general, one can write for Hermitian $R \in \mathbb{C}^{n\times n}$,

$$\nabla_F R = R - FRF^* = GJG^*, \tag{1.3.3}$$

for some triangular $F \in \mathbb{C}^{n\times n}$, a signature matrix $J = (I_p \oplus -I_q) \in \mathbb{C}^{r\times r}$, and $G \in \mathbb{C}^{n\times r}$, with r independent of n. The pair $\{G, J\}$ will be called a ∇_F-generator of R. Because Toeplitz and, as we shall see later, several Toeplitz-related matrices are best studied via this definition, matrices with low ∇_F-displacement rank will be called *Toeplitz-like*. However, this is strictly a matter of convenience.

We can also consider non-Hermitian matrices R, in which case the displacement can be defined as

$$\nabla_{F,A}R \overset{\Delta}{=} R - FRA^*, \tag{1.3.4}$$

where F and A are $n \times n$ lower triangular matrices. In some cases, F and A may coincide—see (1.3.7) below. When $\nabla_{F,A}R$ has low rank, say, r, we can factor it (nonuniquely) as

$$\nabla_{F,A}R = GB^*, \tag{1.3.5}$$

where G and B are also called generator matrices,

$$R - FRA^* = GB^*. \tag{1.3.6}$$

One particular example is the case of a non-Hermitian Toeplitz matrix $T = [c_{i-j}]_{i,j=0}^{n-1}$, which can be seen to satisfy

$$T - ZTZ^* = \begin{bmatrix} c_0 & 1 \\ c_1 & 0 \\ \vdots & \vdots \\ c_{n-1} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & c_{-1}^* \\ \vdots & \vdots \\ 0 & c_{-n+1}^* \end{bmatrix}^* . \tag{1.3.7}$$

This is a special case of (1.3.6) with $F = A = Z$.

A second example is a Vandermonde matrix,

$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \dots & \alpha_n^{n-1} \end{bmatrix}, \tag{1.3.8}$$

which can be seen to satisfy

$$\nabla_{F,Z}V = V - FVZ^* = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}, \tag{1.3.9}$$

where F is now the diagonal matrix $F = \text{diag}\,\{\alpha_1, \dots, \alpha_n\}$.

Another common form of displacement structure, first introduced by Heinig and Rost [HR84], is what we call, again strictly for convenience, a Hankel-like structure. We shall say that a matrix $R \in \mathbb{C}^{n\times n}$ is Hankel-like if it satisfies a displacement equation of the form

$$FR - RA^* = GB^*, \tag{1.3.10}$$

for some lower triangular $F \in \mathbb{C}^{n\times n}$ and $A \in \mathbb{C}^{n\times n}$, and generator matrices $G \in \mathbb{C}^{n\times r}$ and $B \in \mathbb{C}^{n\times r}$, with r independent of n. When R is Hermitian, it is more convenient to express the displacement equation as

$$FR + RF^* = GJG^* \tag{1.3.11}$$

for some generator matrix $G \in \mathbb{C}^{n\times r}$ and signature matrix J that satisfies $J = J^*$ and $J^2 = I$. To avoid a notation explosion, we shall occasionally use the notation $\nabla_{F,A}R$ for both Toeplitz-like and Hankel-like structures.

As an illustration, consider a Hankel matrix, which is a symmetric matrix with real constant entries along the antidiagonals,

$$H = \begin{bmatrix} h_0 & h_1 & h_2 & \dots & h_{n-1} \\ h_1 & h_2 & & \cdot & h_n \\ h_2 & & \cdot & & \vdots \\ \vdots & \cdot & & & \vdots \\ h_{n-1} & h_n & & \dots & h_{2n-1} \end{bmatrix}. \tag{1.3.12}$$

It can be verified that the difference $ZH - HZ^*$ has rank 2 since

$$\imath(ZH - HZ^*) = \imath \cdot \begin{bmatrix} 0 & -h_0 & -h_1 & \dots & -h_{n-2} \\ h_0 & & & & \\ h_1 & & O & & \\ \vdots & & & & \\ h_{n-2} & & & & \end{bmatrix}, \tag{1.3.13}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & h_0 \\ 0 & h_1 \\ \vdots & \vdots \\ 0 & h_{n-2} \end{bmatrix} \underbrace{\begin{bmatrix} 0 & -\imath \\ \imath & 0 \end{bmatrix}}_{J} \begin{bmatrix} 1 & 0 \\ 0 & h_0 \\ 0 & h_1 \\ \vdots & \vdots \\ 0 & h_{n-2} \end{bmatrix}^T.$$

We therefore say that H has displacement rank 2 with respect to the displacement operation (1.3.11) with $F = \imath Z$ and J as above. Here, $\imath = \sqrt{-1}$ and is introduced in order to obtain a J that satisfies the normalization conditions $J = J^*$, $J^2 = I$.

A problem that arises here is that H cannot be fully recovered from its displacement representation, because the entries $\{h_{n-1}, \dots, h_{2n-2}\}$ do not appear in (1.3.14). This "difficulty" can be accommodated in various ways (see, e.g., [CK91b], [HR84], [KS95a]). One way is to border H with zeros and then form the displacement, which will now have rank 4. Another method is to form the $2n \times 2n$ (triangular) Hankel matrix with top row $\{h_0, \dots, h_{2n-1}\}$; now the displacement rank will be two; however, note that in both cases the generators have the same number of entries. In general, the problem is that the displacement equation does not have a unique solution. This will happen when the displacement operator in (1.3.3)–(1.3.6) or (1.3.10)–(1.3.11) has a nontrivial nullspace or kernel. In this case, the generator has to be supplemented by some additional information, which varies from case to case. A detailed discussion, with several examples, is given in [KO96], [KO98].

Other examples of Hankel-like structures include Loewner matrices, Cauchy matrices, and Cauchy-like matrices, encountered, for example, in the study of unconstrained rational interpolation problems (see, e.g., [AA86], [Fie85], [Vav91]). The entries of an $n \times n$ Cauchy-like matrix R have the form

$$C_l = \left[\frac{u_i v_j^*}{f_i - a_j^*}\right]_{i,j=0}^{n-1},$$

where u_i and v_j denote $1\times r$ row vectors and the $\{f_i, a_i\}$ are scalars. The Loewner matrix is a special Cauchy-like matrix that corresponds to the choices $r = 2$, $u_i = \begin{bmatrix} \beta_i & 1 \end{bmatrix}$, and $v_i = \begin{bmatrix} 1 & -\kappa_i \end{bmatrix}$, and, consequently, $u_i v_j^* = \beta_i - \kappa_j^*$:

$$L_w = \left[\frac{\beta_i - \kappa_j^*}{f_i - a_j^*} \right]_{i,j=0}^{n-1}.$$

Cauchy matrices, on the other hand, arise from the choices $r = 1$ and $u_i = 1 = v_i$:

$$C = \left[\frac{1}{f_i - a_j^*} \right]_{i,j=0}^{n-1}.$$

It is easy to verify that a Cauchy-like matrix is Hankel-like since it satisfies a displacement equation of the form

$$FC_l - C_l A^* = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-1} \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix}^*, \tag{1.3.14}$$

where F and A are diagonal matrices:

$$F = \text{diagonal}\,\{f_0, \ldots, f_{n-1}\}, \quad A = \text{diagonal}\,\{a_0, \ldots, a_{n-1}\}.$$

Hence, Loewner and Cauchy matrices are also Hankel-like. Another simple example is the Vandermonde matrix (1.3.8) itself, since it satisfies not only (1.3.9) but also

$$AV - VZ^* = \begin{bmatrix} \frac{1}{\alpha_1} \\ \frac{1}{\alpha_2} \\ \vdots \\ \frac{1}{\alpha_n} \end{bmatrix} \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix}, \tag{1.3.15}$$

where A is the diagonal matrix (assuming $\alpha_i \neq 0$)

$$A = \text{diagonal}\left\{ \frac{1}{\alpha_1}, \ldots, \frac{1}{\alpha_n} \right\}.$$

Clearly, the distinction between Toeplitz-like and Hankel-like structures is not very tight, since many matrices can have both kinds of structure including Toeplitz matrices themselves (cf. (1.2.5) and (1.3.1)).

Toeplitz- and Hankel-like structures can be regarded as special cases of the generalized displacement structure [KS91], [Say92], [SK95a], [KS95a]:

$$\Omega R \Delta^* - FRA^* = GB^*, \tag{1.3.16}$$

where $\{\Omega, \Delta, F, A\}$ are $n \times n$ and $\{G, B\}$ are $n \times r$. Such equations uniquely define R when the diagonal entries $\{\omega_i, \delta_i, f_i, a_i\}$ of the displacement operators $\{\Omega, \Delta, F, A\}$ satisfy

$$\omega_i \delta_j^* - f_i a_j^* \neq 0 \quad \text{for all } i, j. \tag{1.3.17}$$

This explains the difficulty we had in the Hankel case, where the diagonal entries of $F = A = Z$ in (1.3.14) violate the above condition. The restriction that $\{\Omega, \Delta, F, A\}$ are lower triangular is the most general one that allows recursive triangular factorization (cf. a result in [LK86]). As mentioned earlier, since this is a critical feature in most of our applications, we shall assume this henceforth.

The Generating Function Formulation

We may remark that when $\{\Omega, \Delta, F, A\}$ are lower triangular Toeplitz, we can use generating function notation—see [LK84], [LK86]; these can be extended to more general $\{\Omega, \Delta, F, A\}$ by using divided difference matrices—see [Lev83], [Lev97]. The generating function formulation enables connections to be made with complex function theory and especially with the extensive theory of reproducing kernel Hilbert spaces of entire functions (deBranges spaces)—see, e.g., [AD86], [Dym89b], [AD92].

Let us briefly illustrate this for the special cases of Toeplitz and Hankel matrices, $T = [c_{i-j}]$, $H = [h_{i+j}]$. To use the generating function language, we assume that the matrices are semi-infinite, i.e., $i, j \in [0, \infty)$. Then straightforward calculation will yield, assuming $c_0 = 1$, the expression

$$T(z,w) \triangleq \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} c_{i-j} z^i w^{*j} = \frac{c(z) + c^*(w)}{2(1 - zw^*)}, \tag{1.3.18}$$

where $c(z)$ is (a so-called Carathéodory function)

$$c(z) = 1 + 2\sum_{i=1}^{\infty} c_j z^j.$$

The expression can also be rewritten as

$$T(z,w) = \frac{G_1(z) J_1 G_1^*(w)}{d_1(z,w)},$$

where

$$G_1(z) = \begin{bmatrix} \frac{c(z)+1}{2} & \frac{c(z)-1}{2} \end{bmatrix}, \quad J_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad d_1(z,w) = 1 - zw^*.$$

In the Hankel case, we can write

$$H(z,w) \triangleq \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} h_{i+1} z^i w^{*j} = \frac{zh(z) - w^* h^*(w)}{\imath(z - w^*)} = \frac{G_2(z) J_2 G_2^*(w)}{d_2(z,w)},$$

where

$$G_2(z) = \begin{bmatrix} 1 & zh(z) \end{bmatrix}, \quad h(z) = \sum_{j=0}^{\infty} h_j z^j,$$

and

$$d_2(z) = \imath(z - w^*), \quad J = \begin{bmatrix} 0 & -\imath \\ \imath & 0 \end{bmatrix}, \quad \imath = \sqrt{-1}\,.$$

Generalizations can be obtained by using more complex $\{G(\cdot), J\}$ matrices and with

$$d(z,w) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} d_{ij} z^i w^{*j}.$$

However, to admit recursive triangular factorization, one must asume that $d(z,w)$ has the form (see [LK86])

$$d(z,w) = a(z)a^*(w) - b(z)b^*(w)$$

for some $\{a(z), b(z)\}$. The choice of $d(z,w)$ also has a geometric significance. For example, $d_1(z,w)$ partitions the complex plane with respect to the unit circle, as follows:

$$\Omega_o = \{z : \ 1 - |z|^2 = 0\}\ , \quad \Omega_+ = \{z : \ |z| > 1\}\ , \quad \Omega_- = \{z : \ |z| < 1\}\ .$$

Similarly, $d_2(z,w)$ partitions the plane with respect to the real axis. If we used $d_3(z,w) = z + w^*$, we would partition the plane with respect to the imaginary axis.

We may also note that the matrix forms of

$$d_1(z,w)R(z,w) = G_1(z)JG_1^*(w)$$

will be, in an obvious notation,

$$\mathcal{R} - \mathcal{Z}\mathcal{R}\mathcal{Z}^* = \mathcal{G}_1 J_1 \mathcal{G}_1^*, \tag{1.3.19}$$

while using $d_2(z,w)$ it will be

$$\imath(\mathcal{Z}\mathcal{R} - \mathcal{R}\mathcal{Z}^*) = \mathcal{G}_2 J_2 \mathcal{G}_2^*. \tag{1.3.20}$$

Here, $\mathcal{Z}$ denotes the semi-infinite lower triangular shift matrix. Likewise, $\{\mathcal{R}, \mathcal{G}_1, \mathcal{G}_2\}$ denote semi-infinite matrices.

We shall not pursue the generating function descriptions further here. They are useful, inter alia, for studying root distribution problems (see, e.g., [LBK91]) and, as mentioned above, for making connections with the mathematical literature, especially the Russian school of operator theory. A minor reason for introducing these descriptions here is that they further highlight connections between displacement structure theory and the study of discrete-time and continuous-time systems, as we now explain briefly.

Lyapunov, Stein, and Displacement Equations

When $J = I$, (1.3.19) and (1.3.20) are the discrete-time and continuous-time Lyapunov equations much studied in system theory, where the association between discrete-time systems and the unit circle and continuous-time systems and half-planes, is well known. There are also well-known transformations (see [Kai80, p. 180])between discrete-time and continuous-time (state-space) systems, so that in principle all results for Toeplitz-like displacement operators can be converted into the appropriate results for Hankel-like operators. This is one reason that we shall largely restrict ourselves here to the Toeplitz-like Hermitian structure (1.3.3); more general results can be found in [KS95a].

A further remark is that equations of the form (1.3.10), but with general right sides, are sometimes called Sylvester equations, while those of the form (1.3.6) are called Stein equations. For our studies, low-rank factorizations of the right side as in (1.3.10) and (1.3.6) and especially (1.3.11) and (1.3.3) are critical (as we shall illustrate immediately), which is why we call these special forms *displacement equations*.

Finally, as we shall briefly note in Sec. 1.14.1, there is an even more general version of displacement theory applying to "time-variant" matrices (see, e.g., [SCK94], [SLK94b], [CSK95]). These extensions are useful, for example, in adaptive filtering applications and also in matrix completion problems and interpolation problems, where matrices change with time but in such a way that certain displacement differences undergo only low-rank variations.

A Summary

To summarize, there are several ways to characterize the structure of a matrix, using for example (1.3.6), (1.3.10), or (1.3.16). However, in all cases, the main idea is to describe an $n \times n$ matrix R more compactly by $n \times r$ generator matrices $\{G, B\}$, with $r \ll n$. Since the generators have $2rn$ entries, as compared to n^2 entries in R, a computational gain of one order of magnitude can in general be expected from algorithms that operate on the generators directly.

The implications of this fact turn out to be far reaching and have connections with many other areas; see, e.g., [KS95a] and the references therein. In this chapter, we focus mainly on results that are relevant to matrix computations and that are also of interest to the discussions in the later chapters.

The first part of our presentation focuses exclusively on strongly regular Hermitian Toeplitz-like matrices, viz., those that satisfy

$$R - FRF^* = GJG^*, \quad J = J^*, \quad J^2 = I, \tag{1.3.21}$$

for some full rank $n \times r$ matrix G, with $r \ll n$, and lower triangular F. We also assume that the diagonal entries of F satisfy

$$1 - f_i f_j^* \neq 0 \quad \text{for all } i, j \tag{1.3.22}$$

so that (1.3.21) defines R uniquely. Once the main ideas have been presented for this case, we shall then briefly state the results for non-Hermitian Toeplitz-like and Hankel-like matrices. A more detailed exposition for these latter cases, and for generalized displacement equations (1.3.16), can be found in [Say92], [KS95a], [SK95a].

1.4 APPLICATION TO THE FAST EVALUATION OF MATRIX-VECTOR PRODUCTS

The evaluation of matrix-vector products is an important ingredient of several fast algorithms (see, e.g., Chs. 5 and 8), and we discuss it briefly here.

Consider an $n \times n$ Hermitian matrix R with (Toeplitz-like) displacement generator $\{Z, G, J\}$ as in (1.2.4). Given G, we can deduce an explicit representation for R in terms of the columns of G. Indeed, using the fact that Z is a nilpotent matrix, viz., $Z^n = 0$, we can check that the unique solution of (1.2.4) is

$$R = \sum_{i=0}^{n-1} Z^i G J G^* Z^{*i}. \tag{1.4.1}$$

Let us partition the columns of G into two sets $\{x_i\}_{i=0}^{p-1}$ and $\{y_i\}_{i=0}^{q-1}$,

$$G \triangleq \begin{bmatrix} x_0 & x_1 & \dots & x_{p-1} & y_0 & y_1 & \dots & y_{q-1} \end{bmatrix}, \quad p + q = r.$$

It is then easy to see that (1.4.1) is equivalent to the representation

$$R = \sum_{i=0}^{p-1} \mathcal{L}(x_i)\mathcal{L}^*(x_i) - \sum_{i=0}^{q-1} \mathcal{L}(y_i)\mathcal{L}^*(y_i), \tag{1.4.2}$$

where the notation $\mathcal{L}(x)$ denotes a lower triangular Toeplitz matrix whose first column is x, e.g.,

$$\mathcal{L}\left(\begin{bmatrix} a \\ b \\ c \end{bmatrix}\right) = \begin{bmatrix} a & 0 & 0 \\ b & a & 0 \\ c & b & a \end{bmatrix}.$$

Formula (1.4.2) expresses matrices R with displacement structure (1.2.4) in terms of products of triangular Toeplitz matrices. The special cases of Toeplitz matrices and their inverses and products are nice examples.

Among other applications (see, e.g., [KVM78]), the representation (1.4.2) can be exploited to speed up matrix-vector products of the form Ra for any column vector a. In general, such products require $O(n^2)$ operations. Using (1.4.2), the computational cost can be reduced to $O(rn\log 2n)$ by using the fast Fourier transform (FFT) technique. This is because, in view of (1.4.2), evaluating the product Ra involves evaluating products of lower or upper triangular *Toeplitz* matrices by vectors, which is equivalent to a convolution operation. For related applications, see [BP94], [GO94c], and Ch. 5.

1.5 TWO FUNDAMENTAL PROPERTIES

Two fundamental *invariance* properties that underlie displacement structure theory are

(a) invariance of displacement structure under inversion and

(b) invariance of displacement structure under Schur complementation.

Lemma 1.5.1 (Inversion). *If R is an $n\times n$ invertible matrix that satisfies* (1.3.21) *for some full rank $G\in\mathbb{C}^{n\times r}$, then there must exist a full rank matrix $H\in C^{r\times n}$ such that*

$$R^{-1}-F^*R^{-1}F=H^*JH\,. \tag{1.5.1}$$

Proof: The block matrix

$$\begin{bmatrix} R & F \\ F^* & R^{-1}\end{bmatrix}$$

admits the following block triangular decompositions (cf. App. A):

$$\begin{aligned}\begin{bmatrix} R & F \\ F^* & R^{-1}\end{bmatrix} &= \begin{bmatrix} I & 0 \\ F^*R^{-1} & I\end{bmatrix}\begin{bmatrix} R & 0 \\ 0 & R^{-1}-F^*R^{-1}F\end{bmatrix}\begin{bmatrix} I & 0 \\ F^*R^{-1} & I\end{bmatrix}^* \\ &= \begin{bmatrix} I & FR \\ 0 & I\end{bmatrix}\begin{bmatrix} R-FRF^* & 0 \\ 0 & R^{-1}\end{bmatrix}\begin{bmatrix} I & FR \\ 0 & I\end{bmatrix}^*.\end{aligned}$$

Now Sylvester's law of inertia (see also App. A) implies that

$$\text{Inertia}\{R^{-1}-F^*R^{-1}F\} = \text{Inertia}\{R-FRF^*\}.$$

It follows from (1.3.21) that there must exist a full rank $r\times n$ matrix H such that (1.5.1) is valid.

$\diamond$

An immediate application of the above result is to verify that the inverse of a Toeplitz matrix also has displacement structure. Indeed, we know from (1.2.5) that for a Hermitian Toeplitz matrix T, Inertia $(T-ZTZ^*)=(1,1)$. It then follows from Lemma 1.5.1 that the inertia of $(T^{-1}-Z^*T^{-1}Z)$ is also $(1,1)$. But $\tilde{I}T^{-1}\tilde{I}=T^{-*}$ (since $\tilde{I}T\tilde{I}=T^*$) and $\tilde{I}Z^*\tilde{I}=Z$, where $\tilde{I}$ is the reverse identity matrix with ones on the antidiagonal and zeros elsewhere. Hence, Inertia $(T^{-*}-ZT^{-*}Z^*)=(1,1)$, which shows that $T^{-*}-ZT^{-*}Z^*$ has rank 2 with one positive signature and one negative signature. This discussion underlies the famous Gohberg–Semencul formula.

It is worth noting that the result of Lemma 1.5.1 requires no special assumptions (e.g., triangularity) on the matrix F.

The second striking result of displacement structure theory is the following, first stated deliberately in vague terms:

The Schur complements of a structured matrix R inherit its displacement structure. Moreover, a so-called generalized Schur algorithm yields generator matrices for the Schur complements.

In this way, we can justify the low displacement rank property of the Toeplitz matrix combinations that we listed before in the introduction of Sec. 1.2, viz., T_1T_2, $T_1 - T_2T_3^{-1}T_4$, and $(T_1T_2)^{-1}T_3$. Assuming for simplicity square matrices $\{T_1, T_2, T_3\}$, we note that these combinations are Schur complements of the following extended matrices, all of which have low displacement ranks (for suitable choices of F):

$$\begin{bmatrix} -T & I \\ I & 0 \end{bmatrix}, \quad \begin{bmatrix} -I & T_2 \\ T_1 & 0 \end{bmatrix}, \quad \begin{bmatrix} T_3 & T_4 \\ T_2 & T_1 \end{bmatrix}, \quad \begin{bmatrix} I & T_2 & 0 \\ T_1 & 0 & T_3 \\ 0 & I & 0 \end{bmatrix}. \tag{1.5.2}$$

More formally, the result reads as follows.

Lemma 1.5.2 (Schur Complementation). *Consider $n \times n$ matrices R and F. Assume that F is block lower triangular (F_1 and F_3 need not be triangular),*

$$F = \begin{bmatrix} F_1 & 0 \\ F_2 & F_3 \end{bmatrix},$$

partition R accordingly with F,

$$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix},$$

and assume that R_{11} is invertible. Then

$$\text{rank}\,(R_{11} - F_1R_{11}F_1^*) \leq \text{rank}\,(R - FRF^*), \tag{1.5.3}$$

$$\text{rank}\,(\Delta - F_3\Delta F_3^*) \leq \text{rank}\,(R - FRF^*), \tag{1.5.4}$$

where $\Delta = R_{22} - R_{21}R_{11}^{-1}R_{12}$.

Proof: The first inequality follows immediately since R_{11} is a submatrix of R. For the second inequality we first note that

$$\text{rank}\,(R^{-1} - F^*R^{-1}F) = \text{rank}\,(R - FRF^*).$$

We now invoke a block matrix formula for R^{-1} (cf. App. A),

$$R^{-1} = \begin{bmatrix} R_{11}^{-1} + E\Delta^{-1}P & -E\Delta^{-1} \\ \Delta^{-1}P & \Delta^{-1} \end{bmatrix}, \quad E = R_{11}^{-1}R_{12}, \quad P = R_{21}R_{11}^{-1},$$

and observe that Δ^{-1} is a submatrix of R^{-1}. Hence,

$$\text{rank}\,(\Delta^{-1} - F_3^*\Delta^{-1}F_3) \leq \text{rank}\,(R^{-1} - F^*R^{-1}F).$$

But by the first result in the lemma we have

$$\text{rank}\,(\Delta - F_3\Delta F_3^*) = \text{rank}\,(\Delta^{-1} - F_3^*\Delta^{-1}F_3).$$

We thus conclude that rank $(\Delta - F_3\Delta F_3^*) \leq$ rank $(R - FRF^*)$.

◇

Hence, it follows from the statement of the lemma that if R has low displacement rank with respect to the displacement $R - FRF^*$, then its Schur complement Δ has low displacement rank with respect to the displacement $\Delta - F_3\Delta F_3^*$. This result for $F = Z$ was first noted in [Mor80], and in its extended form it was further developed in the Ph.D. research of Delosme [Del82], Lev-Ari [Lev83], Chun [Chu89], Pal [Pal90], Ackner [Ack91], and Sayed [Say92].

1.6 FAST FACTORIZATION OF STRUCTURED MATRICES

The concept of Schur complements perhaps first arose in the context of triangular factorization of matrices. It was implicit in Gauss's work on linear equations and more explicit in the remarkable paper of Schur [Sch17]. In this section we consider afresh the problem of triangular factorization, which is basically effected by the Gaussian elimination technique, and show that adding displacement structure allows us to speed up the Gaussian elimination procedure. This will lead us to what we have called generalized Schur algorithms. We shall develop them here for the displacement structure (1.3.21) and later state variations that apply to non-Hermitian Toeplitz-like and also Hankel-like structures. More details on these latter cases can be found in [KS95a]. Our study of the special case (1.3.21) will be enough to convey the main ideas.

We start by reviewing what we shall call the Gauss–Schur reduction procedure.

1.6.1 Schur Reduction (Deflation)

The triangular decomposition of a matrix $R \in \mathbb{C}^{n\times n}$ will be denoted by

$$R = L D^{-1} L^* , \tag{1.6.1}$$

where $D = \text{diag}\{d_0, d_1, \ldots, d_{n-1}\}$ is a diagonal matrix and the lower triangular factor L is normalized in such a way that the $\{d_i\}$ appear on its main diagonal. The nonzero part of the consecutive columns of L will be denoted by l_i. They can be obtained recursively as follows.

Algorithm 1.6.1 (Schur Reduction). *Given $R \in \mathbb{C}^{n\times n}$, start with $R_0 = R$ and repeat for $i = 0, 1, \ldots, n-1$:*

1. *Let l_i denote the first column of R_i and d_i denote the upper-left-corner element of R_i.*

2. *Perform the Schur reduction step:*

$$\begin{bmatrix} 0 & 0 \\ 0 & R_{i+1} \end{bmatrix} = R_i - l_i d_i^{-1} l_i^* . \tag{1.6.2}$$

The matrix R_i is known as the Schur complement of the leading $i \times i$ block of R.

◇

We therefore see that the Schur reduction step (1.6.2) *deflates* the matrix R_i by subtracting a rank 1 matrix from it and leads to a new matrix R_{i+1} that has one less row and one less column than R_i.

By successively repeating (1.6.2) we obtain the triangular factorization of R,

$$\begin{aligned} R &= l_0 d_0^{-1} l_0^* + \begin{bmatrix} 0 & 0 \\ 0 & R_1 \end{bmatrix}, \quad R_1 \in \mathbb{C}^{n-1 \times n-1} \\ &= l_0 d_0^{-1} l_0^* + \begin{bmatrix} 0 \\ l_1 \end{bmatrix} d_1^{-1} \begin{bmatrix} 0 \\ l_1 \end{bmatrix}^* + \begin{bmatrix} 0 & 0 \\ 0 & R_2 \end{bmatrix}, \quad R_2 \in \mathbb{C}^{n-2 \times n-2} \\ &\vdots \\ &= LD^{-1}L^*. \end{aligned}$$

It is also easy to verify that a suitable partitioning of L and D provides triangular decompositions for the leading principal block of R and its Schur complement.

Lemma 1.6.1 (Partitioning of the Triangular Decomposition). *Assume that R, L, and D are partitioned as*

$$R = \begin{array}{c} \overbrace{\quad}^{i} \; \overbrace{\quad}^{n-i} \\ \begin{bmatrix} P_i & Q_i^* \\ Q_i & S_i \end{bmatrix} \end{array} \begin{array}{l} \}i \\ \}n-i \end{array}, \quad L = \begin{array}{c} \overbrace{\quad}^{i} \; \overbrace{\quad}^{n-i} \\ \begin{bmatrix} \bar{L}_i & 0 \\ \tilde{L}_i & L_i \end{bmatrix} \end{array} \begin{array}{l} \}i \\ \}n-i \end{array}, \quad D = \begin{array}{c} \overbrace{\quad}^{i} \; \overbrace{\quad}^{n-i} \\ \begin{bmatrix} \bar{D}_i & 0 \\ 0 & D_i \end{bmatrix} \end{array} \begin{array}{l} \}i \\ \}n-i \end{array}.$$

Then the leading principal block P_i and its Schur complement $R_i = S_i - Q_i P_i^{-1} Q_i^*$ admit the following triangular decompositions:

$$P_i = \bar{L}_i \bar{D}_i^{-1} \bar{L}_i^* \quad \text{and} \quad R_i = L_i D_i^{-1} L_i^* . \tag{1.6.3}$$

◇

1.6.2 Close Relation to Gaussian Elimination

The Schur reduction procedure is in fact the same as Gaussian elimination. To see this, consider the first step of (1.6.2):

$$\begin{bmatrix} 0 & 0 \\ 0 & R_1 \end{bmatrix} = R - l_0 d_0^{-1} l_0^* . \tag{1.6.4}$$

If we partition the entries of l_0 into $l_0 = \text{col}\{d_0, t_0\}$, where t_0 is also a column vector, then the above equality can be written as

$$R = \begin{bmatrix} 1 & 0 \\ t_0 d_0^{-1} & I_{n-1} \end{bmatrix} \begin{bmatrix} d_0 & \\ & R_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_0 d_0^{-1} & I_{n-1} \end{bmatrix}^* \tag{1.6.5}$$

or, equivalently, as

$$\begin{bmatrix} 1 & 0 \\ -t_0 d_0^{-1} & I_{n-1} \end{bmatrix} R = \begin{bmatrix} d_0 & 0 \\ t_0 & R_1 \end{bmatrix}^*,$$

where I_{n-1} is the identity matrix of dimension $(n-1)$. This relation shows why (1.6.2), which we called Schur reduction, is closely related to Gaussian elimination. Schur reduction goes more explicitly toward matrix factorization. Note also that the above Schur reduction procedure can readily be extended to strongly regular non-Hermitian matrices to yield the so-called LDU decompositions (see, e.g., [KS95a]).

1.6.3 A Generalized Schur Algorithm

In general, Alg. 1.6.1 requires $\mathcal{O}(n^3)$ operations to factor R. However, when R has displacement structure, the computational burden can be significantly reduced by exploiting the fact that the Schur complements R_i all inherit the displacement structure of R.

Schur algorithms can be stated in two different, but equivalent, forms—via a set of equations or in a less traditional form as what we call an *array algorithm.* In the latter, the key operation is the triangularization by a sequence of elementary unitary or J-unitary operations of a prearray formed from the data at a certain iteration; the information needed to form the prearray for the next iteration can be read out from the entries of the triangularized prearray. No equations, or at most one or two very simple explicit equations, are needed. We first state a form of the algorithm, before presenting the derivation.

Algorithm 1.6.2 (A Generalized Schur Algorithm). *Given a matrix $R \in \mathbb{C}^{n\times n}$ that satisfies* (1.3.21) *and* (1.3.22) *for some full rank $G \in \mathbb{C}^{n\times r}$, start with $G_0 = G$ and perform the following steps for $i=0,\ldots,n-1$:*

1. *Let g_i be the top row of G_i and let F be partitioned as*

$$F = \begin{array}{c} \begin{array}{cc} \overbrace{\quad}^{i} & \overbrace{\quad}^{n-i} \end{array} \\ \left[\begin{array}{cc} \bar{F}_i & 0 \\ \tilde{F}_i & F_i \end{array}\right] \end{array} \begin{array}{l} \\ \}\, i \\ \}\, n-i \end{array}. \tag{1.6.6}$$

That is, F_i is obtained by ignoring the leading i rows and columns of F. Now compute l_i by solving the linear system of equations[3]

$$(I_{n-i} - f_i^* F_i)\, l_i = G_i\, J\, g_i^*. \tag{1.6.7}$$

Define the top element of l_i,

$$d_i = g_i J g_i^* / (1 - f_i f_i^*). \tag{1.6.8}$$

2. *Form the prearray shown below and choose a $(d_i^{-1} \oplus J)$-unitary matrix Σ_i that eliminates the top row of G_i:*

$$\left[\begin{array}{cc} F_i l_i & G_i \end{array}\right] \Sigma_i = \left[\begin{array}{cc} l_i & \begin{array}{c} 0 \\ G_{i+1} \end{array} \end{array}\right]. \tag{1.6.9}$$

This will give us a right-hand side as shown, where the matrix G_{i+1} can be used to repeat steps 1 *and* 2*; a matrix Σ_i is $(d_i^{-1} \oplus J)$-unitary if*

$$\Sigma_i (d_i^{-1} \oplus J) \Sigma_i^* = (d_i^{-1} \oplus J).$$

Notice that G_{i+1} has one less row than G_i.

3. *Then the $\{l_i\}$ define the successive columns of L, $D = \operatorname{diag}\{d_i\}$, and $R = LD^{-1}L^*$.*

◇

[3] F_i is not only triangular, but in many applications it is usually sparse and often diagonal or bidiagonal, so that (1.6.7) is easy to solve and (see (1.6.9)) $F_i l_i$ is easy to form. For example, whenever F is strictly lower triangular, say, $F = Z$ or $F = Z \oplus Z$, then $l_i = G_i J g_i^*$.

Remark 1. Unitary transformations can have several forms, which are discussed in App. B. A graphic depiction of the algorithm will be given in Sec. 1.6.6. When the matrix F is sparse enough (e.g., diagonal or bidiagonal), so that the total number of flops required for solving the linear system (1.6.7) is $\mathcal{O}(n-i)$, then the computational complexity of Alg. 1.6.2 is readily seen to be $\mathcal{O}(rn^2)$.

Remark 2. We shall show in Lemma 1.6.3 that the matrices $\{G_i\}$ that appear in the statement of the algorithm are in fact generator matrices for the successive Schur complements of R, viz.,

$$R_i - F_i G_i F_i^* = G_i J G_i^* \,. \tag{1.6.10}$$

Remark 3 (An Explicit Form). It can be shown that G_{i+1} can be obtained from G_i by an explicit calculation:

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = \left\{ G_i + (\Phi_i - I_{n-i})\, G_i \frac{J g_i^* g_i}{g_i J g_i^*} \right\} \Theta_i \,, \tag{1.6.11}$$

where Θ_i is any J-unitary matrix, and Φ_i is the so-called Blaschke–Potapov matrix

$$\Phi_i = (I_{n-i} - f_i^* F_i)^{-1}(F_i - f_i I_{n-i}) = (F_i - f_i I_{n-i})(I_{n-i} - f_i^* F_i)^{-1} \,. \tag{1.6.12}$$

Remark 4. Although the above statement is for Hermitian matrices that satisfy displacement equations of the form $R - FRF^* = GJG^*$, there are similar algorithms for non-Hermitian Toeplitz-like matrices and also for Hankel-like matrices (see Secs. 1.11 and 1.12). The discussion we provide in what follows for the Hermitian Toeplitz-like case highlights most of the concepts that arise in the study of structured matrices.

Remark 5 (Terminology). The ultimate conclusion is that the above generalized Schur algorithm is the result of combining displacement structure with Gaussian elimination in order to speed up the computations. We have called it *a* (rather than *the*) generalized Schur algorithm because there are many variations that can be obtained by different choices of the matrices $\{\Sigma_i, \Theta_i\}$ *and* for different forms of displacement structure. Generalized Schur algorithms for the general displacement (1.3.16) can be found in [KS91], [KS95a], [SK95a]. Finally, we mention that the classical (1917) Schur algorithm is deduced as a special case in Sec. 1.7.6.

1.6.4 Array Derivation of the Algorithm

The equation forms of the algorithm (i.e., (1.6.7) and (1.6.11)) can be derived in several different ways—see, e.g., [Lev83], [LK84], [LK92], [Say92], [KS95a], and the array-based algorithm deduced from it. Here we shall present a direct derivation of the array algorithm using minimal prior knowledge; in fact, we shall deduce the equation form from the array algorithm. The presentation follows that of [BSLK96], [BKLS98a].

The key matrix result is the next lemma. We first introduce some notation. Recall the factorization (1.6.1),

$$R = LD^{-1}L^*,$$

where L is lower triangular with diagonal entries d_i while $D = \text{diag}\{d_i\}$. Now let us define the upper triangular matrix

$$U \triangleq L^{-*}D \tag{1.6.13}$$

and write

$$R^{-1} = L^{-*}DL^{-1} = (L^{-*}D)D^{-1}(DL^{-1}) = UD^{-1}U^*. \tag{1.6.14}$$

The factorization (1.6.14) is somewhat nontraditional since we use D^{-1} rather than D. That is, the same diagonal factor D^{-1} is used in the factorizations (1.6.1) and (1.6.14) for both R and R^{-1}. The reason for doing this will become clear soon.

Lemma 1.6.2 (Key Array Equation). *Consider a Toeplitz-like strongly regular and Hermitian matrix R with a full rank generator matrix $G \in \mathbb{C}^{n\times r}$, i.e., $R - FRF^* = GJG^*$. Then there must exist a $(D^{-1} \oplus J)$-unitary matrix Ω such that*

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix}\Omega = \begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix}, \quad \Omega(D^{-1}\oplus J)\Omega^* = (D^{-1}\oplus J). \tag{1.6.15}$$

Proof: Recall from Lemma 1.5.1 that there exists a full rank matrix H^* such that

$$R^{-1} - F^*R^{-1}F = H^*JH.$$

Hence, if we reconsider the block matrix

$$\begin{bmatrix} R & F \\ F^* & R^{-1} \end{bmatrix} \tag{1.6.16}$$

that appeared in the proof of Lemma 1.5.1 and use the displacement equations (1.3.21) and (1.5.1), we can rewrite the block triangular factorizations in the proof of the lemma as

$$\begin{bmatrix} R & F \\ F^* & R^{-1} \end{bmatrix} = \begin{bmatrix} FL & G \\ U & 0 \end{bmatrix}\begin{bmatrix} D^{-1} & 0 \\ 0 & J \end{bmatrix}\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix}^* \tag{1.6.17}$$

$$= \begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix}\begin{bmatrix} D^{-1} & 0 \\ 0 & J \end{bmatrix}\begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix}^*. \tag{1.6.18}$$

The center matrix in both (1.6.17) and (1.6.18) is the same diagonal matrix $(D^{-1}\oplus J)$; it was to achieve this that we started with the nontraditional factorizations $R = LD^{-1}L^*$ and $R^{-1} = UD^{-1}U^*$, with D^{-1} in both factorizations.

The next step is to observe that, in view of the invertibility of L and U and the full rank assumption on G and H^*, the matrices

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix}$$

have full rank (equal to $n+r$). It then follows from Lemma A.4.3 in App. A that there must exist a $(D^{-1}\oplus J)$-unitary matrix Ω such that (1.6.15) holds.

◊

We shall focus first on the equality of the first block row in (1.6.15). The second block rows can be used as the basis for the derivation of an efficient algorithm for factoring R^{-1} and for determining H^*. We shall pursue this issue later in Sec. 1.10.

By confining ourselves to the first block row of (1.6.15), we note that there always exists a $(D^{-1}\oplus J)$-unitary matrix Ω that performs the transformation

$$\begin{bmatrix} FL & G \end{bmatrix}\Omega = \begin{bmatrix} L & 0 \end{bmatrix}. \tag{1.6.19}$$

The matrices $\{F, L, J\}$ are uniquely specified by R and the displacement equation $R - FRF^* = GJG^*$. There is flexibility in choosing G since we can also use $G\Theta$ for any J-unitary Θ, i.e., one that obeys $\Theta J\Theta^* = J = \Theta^*J\Theta$.

This freedom allows us to conjecture the following algorithmic consequence of (1.6.19): given any G, form a prearray as shown below in (1.6.20) and triangularize it in any way we wish by some $(D^{-1} \oplus J)$-unitary matrix Ω,

$$\begin{bmatrix} FL & G \end{bmatrix} \Omega = \begin{bmatrix} X & 0 \end{bmatrix}. \qquad (1.6.20)$$

Then we can identify $X = L$. Indeed, by forming the $(D^{-1} \oplus J)$-"norms" of both sides of (1.6.20), we obtain

$$\begin{bmatrix} FL & G \end{bmatrix} \underbrace{\Omega \begin{bmatrix} D^{-1} & 0 \\ 0 & J \end{bmatrix} \Omega^*}_{D^{-1}\oplus J} \begin{bmatrix} L^*F^* \\ G^* \end{bmatrix} = \begin{bmatrix} X & 0 \end{bmatrix} \begin{bmatrix} D^{-1} & 0 \\ 0 & J \end{bmatrix} \begin{bmatrix} X^* \\ 0 \end{bmatrix}.$$

That is,

$$F\,L\,D^{-1}\,L^*\,F^* + G\,J\,G^* = X\,D^{-1}\,X^*.$$

Hence, it follows from (1.3.21) that $R = XD^{-1}X^*$. But $R = LD^{-1}L^*$, so by uniqueness we must have $X = L$.

Continuing with (1.6.19), at first it is of course difficult to see how (1.6.19) can be used to compute L and D when only F, G, and J are given, since the unknown quantity L appears on both sides of the equation. This apparent difficulty is resolved by proceeding *recursively*.

Thus note that the first column of L (and hence of FL) can be obtained by multiplying the displacement equation by the first unit column vector e_0 from the right,

$$LD^{-1}Le_0 - FLD^{-1}L^*F^*e_0 = GJG^*e_0,$$

or

$$l_0 - Fl_0f_0^* = GJg_0^*, \qquad l_0 = (I - Ff_0^*)^{-1}GJg_0^*,$$

where

$$g_0 = \text{the top row of } G.$$

The inverse exists by our solvability assumption (1.3.22), which ensured that the displacement equation has a unique solution.

Now we can find elementary transformations that successively combine the first column of FL with the columns of G so as to null out the top entries of G, i.e., to null out g_0. From (1.6.19) we see that the resulting postarray must have, say, the form

$$\begin{bmatrix} FL & G \end{bmatrix} \Omega_0 = \begin{bmatrix} l_0 & \begin{matrix} 0 \\ F_1l_1 \end{matrix} & \begin{matrix} 0 \\ G_1 \end{matrix} \end{bmatrix},$$

where F_1 and L_1 are (as defined earlier) obtained by omitting the first columns of F and L, and all the entries of G_1 are known.

Now we can prove that G_1 is such that $R_1 \triangleq L_1D_1^{-1}L_1^*$ obeys

$$R_1 - F_1R_1F_1^* = G_1JG_1^*, \qquad (1.6.21)$$

where $D_1 = \text{diag}\{d_1, \ldots, d_{n-1}\}$. This equation allows us to determine the first column of F_1L_1 and then proceed as above. To prove (1.6.21), it will be convenient to proceed more generally by considering the ith step of the recursion.

For this purpose, we recall the partitioning

$$F = \begin{array}{c} \begin{array}{cc} \overbrace{}^{i} & \overbrace{}^{n-i} \end{array} \\ \left[\begin{array}{cc} \bar{F}_i & 0 \\ \tilde{F}_i & F_i \end{array}\right] \end{array} \begin{array}{l} \}\, i \\ \}\, n-i \end{array}, \qquad L = \begin{array}{c} \begin{array}{cc} \overbrace{}^{i} & \overbrace{}^{n-i} \end{array} \\ \left[\begin{array}{cc} \bar{L}_i & 0 \\ \tilde{L}_i & L_i \end{array}\right] \end{array} \begin{array}{l} \}\, i \\ \}\, n-i \end{array},$$

and therefore partition the pre- and postarrays accordingly:

$$\left[\begin{array}{cc} FL & G \end{array}\right] = \left[\begin{array}{cc|c} \bar{F}_i\bar{L}_i & 0 & \multirow{2}{*}{G} \\ \tilde{F}_i\bar{L}_i + F_i\tilde{L}_i & F_i L_i & \end{array}\right] \tag{1.6.22}$$

and

$$\left[\begin{array}{cc} L & 0 \end{array}\right] = \left[\begin{array}{cc|c} \bar{L}_i & 0 & \multirow{2}{*}{0} \\ \tilde{L}_i & L_i & \end{array}\right]. \tag{1.6.23}$$

After i iterations, the first i columns of the postarray are already computed, while the last $(n-i)$ columns of the prearray have not yet been modified. Therefore, the ith intermediate array must have the following form:

$$\left[\begin{array}{cc} FL & G \end{array}\right] \Omega_0 \Omega_1 \ldots \Omega_{i-1} = \left[\begin{array}{cc|c} \bar{L}_i & 0 & 0 \\ \tilde{L}_i & F_i L_i & G_i \end{array}\right], \tag{1.6.24}$$

where G_i denotes the nontrivial element that appears in the upper-right-corner block.

All our results will now follow by using the fact that the prearray (1.6.22), *the intermediate array* (1.6.24), *and the postarray* (1.6.23) *are all* $(D^{-1} \oplus J)$ *equivalent; i.e., their squares in the* $(D^{-1} \oplus J)$ *"metric" are all equal.*

We first establish that the entry G_i in the intermediate array (1.6.24) is a generator matrix of the (leading) Schur complement R_i.

Lemma 1.6.3 (Structure of Schur Complements). *Let G_i be the matrix shown in* (1.6.24). *Then the Schur complement R_i satisfies the displacement equation*

$$R_i - F_i R_i F_i^* = G_i J G_i^*. \tag{1.6.25}$$

Proof: Since Ω and Ω_i are $(D^{-1} \oplus J\}$-unitary, the second block rows of the postarray (1.6.23) and the intermediate array (1.6.24) must satisfy

$$\left[\tilde{L}_i \; F_iL_i \; G_i\right] (\bar{D}_i^{-1} \oplus D_i^{-1} \oplus J) \left[\tilde{L}_i \; F_iL_i \; G_i\right]^* = \left[\tilde{L}_i \; L_i \; 0\right] (\bar{D}_i^{-1} \oplus D_i^{-1} \oplus J) \left[\tilde{L}_i \; L_i \; 0\right]^*.$$

It follows immediately that

$$F_i \left(L_i D_i^{-1} L_i^*\right) F_i^* + G_i J G_i^* = \left(L_i D_i^{-1} L_i^*\right).$$

But we already know from (1.6.3) that $R_i = L_i D_i^{-1} L_i^*$, so (1.6.25) is established.

◇

From (1.6.25) it is easy to obtain a closed-form expression for l_i. In fact, equating the first columns on both sides of (1.6.25) leads to

$$l_i - F_i f_i^* l_i = G_i J g_i^*, \tag{1.6.26}$$

which is (1.6.7). The solvability condition (1.3.22) ensures that (1.6.7) has a unique solution for l_i.

Now recursion (1.6.9) follows by examining the transformation from the ith to the $(i+1)$th iteration, i.e., at the step that updates the right-hand side of (1.6.24). It can be succinctly described as

$$\left[\begin{array}{c|c} F_i l_i & G_i \end{array}\right]\Sigma_i = \left[\begin{array}{c|c} l_i & \begin{matrix}0\\ G_{i+1}\end{matrix} \end{array}\right], \tag{1.6.27}$$

where the irrelevant columns and rows of the pre- and postarrays (i.e., those rows and columns that remain unchanged) were omitted. Also, Σ_i is a submatrix of Ω_i, and it is $(d_i^{-1} \oplus J)$-unitary. We have argued above in (1.6.25) that the matrix G_{i+1} in (1.6.27) is a generator for the Schur complement R_{i+1}. This completes the proof of the array Alg. 1.6.2.

Explicit Equations

Although not needed for the algorithm, we can pursue the argument a bit further and deduce the explicit updating equation (1.6.11). To do this, we first identify the transformation Σ_i. To do this, note first that Σ_i is $(d_i^{-1} \oplus J)$-unitary, i.e., $\Sigma_i(d_i^{-1} \oplus J)\Sigma_i^* = (d_i^{-1} \oplus J)$. Therefore, the inverse of Σ_i is given by

$$\Sigma_i^{-1} = \begin{bmatrix} d_i^{-1} & \\ & J \end{bmatrix}\Sigma_i^*\begin{bmatrix} d_i & \\ & J \end{bmatrix}.$$

It then follows from the array equation (1.6.27) that

$$\begin{bmatrix} F_i l_i & G_i \end{bmatrix}\begin{bmatrix} d_i^{-1} & \\ & J \end{bmatrix} = \begin{bmatrix} l_i & \begin{matrix}0\\ G_{i+1}\end{matrix} \end{bmatrix}\begin{bmatrix} d_i^{-1} & \\ & J \end{bmatrix}\Sigma_i^*. \tag{1.6.28}$$

If we denote the entries of Σ_i by

$$\Sigma_i = \begin{bmatrix} a_i & b_i \\ c_i & s_i \end{bmatrix}, \tag{1.6.29}$$

where $a_i \in C$, $b_i \in C^{1\times r}$, $c_i \in C^{r\times 1}$, and $s_i \in C^{r\times r}$, we conclude by equating the top row on both sides of (1.6.28) that we must have

$$a_i = f_i^*, \quad c_i = Jg_i^*. \tag{1.6.30}$$

In other words, any $(d_i^{-1} \oplus J)$-transformation Σ_i that achieves (1.6.27) must be such that its entries $\{a_i, c_i\}$ are as above. In order to identify the remaining entries $\{b_i, s_i\}$, we note that in view of the $(d_i^{-1} \oplus J)$-unitarity of Σ_i, the entries $\{a_i, b_i, c_i, s_i\}$ must satisfy

$$\begin{bmatrix} a_i & b_i \\ c_i & s_i \end{bmatrix}\begin{bmatrix} d_i^{-1} & 0 \\ 0 & J \end{bmatrix}\begin{bmatrix} a_i & b_i \\ c_i & s_i \end{bmatrix}^* = \begin{bmatrix} d_i^{-1} & 0 \\ 0 & J \end{bmatrix}. \tag{1.6.31}$$

Lemma 1.6.4 (Identification of Σ_i). *Given $\{f_i, g_i, J\}$, all pairs $\{b_i, s_i\}$ that satisfy (1.6.31) are given by*

$$b_i^* = \Theta_i^* g_i^* d_i^{-1}, \tag{1.6.32}$$

$$s_i^* = \Theta_i^*\left[I - \frac{1+f_i^*}{1-f_i f_i^*} g_i^* d_i^{-1} g_i J \right] \tag{1.6.33}$$

for any J-unitary parameter Θ_i $(\Theta_i J \Theta_i^* = J)$.

Proof: It follows from (1.6.31) that

$$\begin{bmatrix} b_i \\ s_i \end{bmatrix} J \begin{bmatrix} b_i \\ s_i \end{bmatrix}^* = \begin{bmatrix} d_i^{-1}(1-|f_i|^2) & -f_i^* d_i^{-1} g_i J \\ -Jg_i^* d_i^{-1} f_i & J - Jg_i^* d_i^{-1} g_i J \end{bmatrix}.$$

Using $d_i(1-|f_i|^2) = g_i J g_i^*$, we can verify after some algebra that the right-hand side of the above expression can be factored as

$$\begin{bmatrix} b_i \\ s_i \end{bmatrix} J \begin{bmatrix} b_i \\ s_i \end{bmatrix}^* = \begin{bmatrix} d_i^{-1} g_i \\ I - \frac{1+f_i}{1-|f_i|^2} Jg_i^* d_i^{-1} g_i \end{bmatrix} J \begin{bmatrix} d_i^{-1} g_i \\ I - \frac{1+f_i}{1-|f_i|^2} Jg_i^* d_i^{-1} g_i \end{bmatrix}^*.$$

But the matrix $\left[\begin{smallmatrix} b_i \\ s_i \end{smallmatrix}\right]$ is full rank since Σ_i is full rank (due to its $(d_i^{-1} \oplus J)$-unitarity). Likewise, the matrix

$$\begin{bmatrix} d_i^{-1} g_i \\ I - \frac{1+f_i}{1-|f_i|^2} Jg_i^* d_i^{-1} g_i \end{bmatrix}$$

is full rank since otherwise there would exist a nonzero vector x such that

$$\begin{bmatrix} d_i^{-1} g_i \\ I - \frac{1+f_i}{1-|f_i|^2} Jg_i^* d_i^{-1} g_i \end{bmatrix} x = 0.$$

This implies that we must have $g_i x = 0$, which in turn implies from the equality of the second block row that $x = 0$. This contradicts the fact that x is nonzero.

Therefore, in view of the result of Lemma A.4.3, we conclude that there should exist a J-unitary matrix Θ_i such that

$$\begin{bmatrix} b_i \\ s_i \end{bmatrix} = \begin{bmatrix} d_i^{-1} g_i \\ I - \frac{1+f_i}{1-|f_i|^2} Jg_i^* d_i^{-1} g_i \end{bmatrix} \Theta_i,$$

as desired.

◇

Substituting (1.6.30) and (1.6.33) into (1.6.27) yields (1.6.11).

1.6.5 An Elementary Section

A useful remark is to note that in (1.6.27), viz.,

$$\begin{bmatrix} F_i l_i & G_i \end{bmatrix} \begin{bmatrix} a_i & b_i \\ c_i & s_i \end{bmatrix} = \begin{bmatrix} l_i & 0 \\ & G_{i+1} \end{bmatrix},$$

we can regard the transformation Σ_i of (1.6.29) as the system matrix of a first-order state-space linear system; the rows of $\{G_i\}$ and $\{G_{i+1}\}$ can be regarded as inputs and outputs of this system, respectively, and the entries of $\{l_i, F_i l_i\}$ can be regarded as the corresponding current and future states.

Let $\Theta_i(z)$ denote the transfer function of the above linear system (with inputs from the left), viz.,

$$\Theta_i(z) = s_i + c_i(z^{-1} - f_i^*)^{-1} b_i.$$

Using (1.6.32) and (1.6.33), and simple algebra, shows that the above expression collapses to

$$\Theta_i(z) = \left[I_r + (B_i(z) - 1)\frac{Jg_i^* g_i}{g_i J g_i^*} \right] \Theta_i, \tag{1.6.34}$$

where

$$B_i(z) = \frac{z - f_i}{1 - f_i^* z}\,. \tag{1.6.35}$$

We therefore see that each step of the generalized Schur algorithm gives rise to a first-order section $\Theta_i(z)$. Such elementary sections have several useful properties. In particular, note that

$$g_i \Theta_i(f_i) = 0, \tag{1.6.36}$$

which shows that $\Theta_i(z)$ has a transmission zero at f_i along the direction of g_i. This blocking property can be used, for example, to obtain efficient recursive solutions to rational interpolation problems (see, e.g., [SKLC94], [BSK94], [BSK99] and Sec. 1.14.3).

1.6.6 A Simple Example

To see how Alg. 1.6.2 works, we consider a simple example with $n=3$ and $r=2$. In this case the pre- and postarrays in (1.6.19) will have the following zero-patterns:

$$\begin{bmatrix} FL & G \end{bmatrix} = \left[\begin{array}{ccc|cc} * & 0 & 0 & * & * \\ * & * & 0 & * & * \\ * & * & * & * & * \end{array}\right], \qquad \begin{bmatrix} L & 0 \end{bmatrix} = \left[\begin{array}{ccc|cc} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \end{array}\right].$$

Using (1.6.7) for $i = 0$, viz.,

$$(I_n - f_0^* F)l_0 = GJg_0,$$

we can determine the first column l_0 of L and, consequently, the first column of FL. In this way, all the entries of the first column of the prearray are completely known. Also, the last two columns of the prearray are known since they are determined by G.

Hence, the $(1, 2)$ block entry of the prearray (i.e., the top row of G) can be eliminated by pivoting with the top entry of the first column of FL. As a result, the first column of the prearray and its last two columns are linearly combined to yield the intermediate array shown below after the transformation Ω_0. The second and third columns of the prearray remain unchanged:

$$\left[\begin{array}{ccc|cc} \boxed{*} & 0 & 0 & \multicolumn{2}{c}{\boxed{* \quad *}} \\ * & * & 0 & * & * \\ * & * & * & * & * \end{array}\right] \xrightarrow{\Omega_0} \left[\begin{array}{ccc|cc} * & 0 & 0 & 0 & 0 \\ * & \boxed{*} & 0 & \multicolumn{2}{c}{\boxed{* \quad *}} \\ * & * & * & * & * \end{array}\right] \xrightarrow{\Omega_1} \left[\begin{array}{ccc|cc} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ * & * & \boxed{*} & \multicolumn{2}{c}{\boxed{* \quad *}} \end{array}\right] \xrightarrow{\Omega_2} \left[\begin{array}{ccc|cc} * & 0 & 0 & 0 & 0 \\ * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \end{array}\right].$$

We now proceed in a similar fashion to the next step. Using (1.6.7) for $i = 1$, viz.,

$$(I_{n-1} - f_1^* F_1)l_1 = G_1 J g_1,$$

we determine the l_1 and, consequently, the second column of FL. The second transformation Ω_1 can now be performed as shown above to yield G_2 (see below), and so on.

The rectangular boxes mark the entries to be eliminated at each step of the recursion by using elementary $(D^{-1} \oplus J)$-unitary transformations (scaled column permutations, Givens rotations, and Householder projections—see App. B). The square boxes mark the position of the pivot elements. The ultimate result of the recursion is that the $(1, 2)$ block of the prearray is eliminated row by row.

1.7 PROPER FORM OF THE FAST ALGORITHM

A useful feature of the explicit form of the generator recursion (1.6.11) is that different choices of the arbitrary J-unitary matrix Θ_i can be more easily used to obtain different variants of the general algorithm, which can have different domains of usefulness. One of these is the so-called proper form.

This means that Θ_i is chosen so as to eliminate all elements of g_i with the exception of a single pivot element. This pivot has to be in the first p positions when $g_i J g_i^* > 0$ and in the last q positions when $g_i J g_i^* < 0$. (Note that the case $g_i J g_i^* = 0$ is ruled out by the strong regularity assumption on R. This is because $d_i \neq 0$ implies $g_i J g_i^* \neq 0$ by (1.6.8).)

1.7.1 Positive Lengths

When $g_i J g_i^* > 0$ holds, we can choose a J-unitary rotation Θ_i that would reduce g_i to the form

$$g_i\Theta_i = \begin{bmatrix} \delta_i & 0 & \dots & 0 \end{bmatrix}, \qquad |\delta_i|^2 = g_i J g_i^*, \tag{1.7.1}$$

where we have chosen, without loss of generality, the nonzero entry to be in the leading position of the postarray. With this particular choice, the generator recursion (1.6.11) can be seen to collapse to

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = G_i\,\Theta_i \begin{bmatrix} 0 & 0 \\ 0 & I_{r-1} \end{bmatrix} + \Phi_i\, G_i\, \Theta_i \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}. \tag{1.7.2}$$

Likewise, the expressions (1.6.7) and (1.6.8) for l_i and d_i become

$$l_i = \delta_i^*(I_{n-i} - f_i^* F_i)^{-1} G_i \Theta_i \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad d_i = |\delta_i|^2/(1 - |f_i|^2). \tag{1.7.3}$$

Equation (1.7.2) yields the following simple statement for steps with $g_i J g_i^* > 0$:

1. Transform G_i into proper form with respect to its first column by using a J-unitary rotation Θ_i.

2. Multiply the first column by Φ_i and keep the rest of the columns unaltered.

The first step eliminates all entries in the first row of G_i except the pivot entry in the upper-left-corner position (see Fig. 1.2 below). The second step is needed to eliminate the pivot entry.

$$G_i = \begin{bmatrix} * & * & * \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} \xrightarrow{\text{Step 1}} \begin{bmatrix} * & 0 & 0 \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} \xrightarrow{\text{Step 2}} \begin{bmatrix} 0 & 0 & 0 \\ * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \end{bmatrix} = \begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix}$$

Figure 1.2. Illustrating the proper form of the generator recursion.

Each step of (1.7.2) can also be depicted graphically as a cascade network of elementary sections, one of which is shown in Fig. 1.3; Θ_i is any J-unitary matrix that rotates

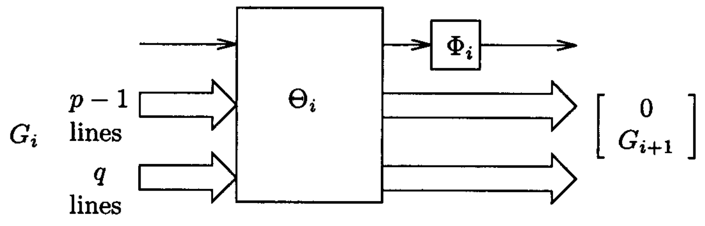

Figure 1.3. Proper form when $g_iJg_i^* > 0$.

the first row of the ith generator to $\begin{bmatrix} \delta_i & 0 \end{bmatrix}$. The rows of G_i enter the section one row at a time. The leftmost entry of each row is applied through the top line, while the remaining entries are applied through the bottom lines. The Blaschke–Potapov matrix Φ_i then acts on the entries of the top line. When $F_i = Z$, the lower shift matrix Φ_i collapses to $\Phi_i = Z$, a delay unit (see the discussion further ahead on shift-structured matrices). In general, note that the first row of each Φ_i is zero, and in this sense Φ_i acts as a generalized delay element. To clarify this, observe that when the entries of the first row of G_i are processed by Θ_i and Φ_i, the values of the outputs of the section will all be zero. The rows of G_{i+1} will start appearing at these outputs only when the second and higher rows of G_i are processed by the section.

1.7.2 Negative Lengths

A similar derivation holds for steps with $g_iJg_i^* < 0$. Now we choose a J-unitary matrix Θ_i so that

$$g_i\Theta_i = \begin{bmatrix} 0 & \ldots & 0 & \delta_i \end{bmatrix}, \qquad |\delta_i|^2 = -g_iJg_i^*, \tag{1.7.4}$$

where the nonzero entry in the postarray is chosen, again for convenience, to be at its last position.

Equation (1.6.11) now collapses to

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = G_i\,\Theta_i \begin{bmatrix} I_{r-1} & 0 \\ 0 & 0 \end{bmatrix} + \Phi_i\,G_i\,\Theta_i \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \tag{1.7.5}$$

Also, expressions (1.6.7) and (1.6.8) for l_i and d_i become

$$l_i = -\delta_i^*(I_{n-i} - f_i^*F_i)^{-1}G_i\Theta_i \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad d_i = |\delta_i|^2/(1-|f_i|^2). \tag{1.7.6}$$

Equation (1.7.5) admits the following simple interpretation:

1. Transform G_i into proper form with respect to its last column by using a J-unitary rotation Θ_i.

2. Multiply the last column by Φ_i and keep the rest of the columns unaltered.

1.7.3 Statement of the Algorithm in Proper Form

We collect the above results into the following statement.

Algorithm 1.7.1 (Fast Algorithm in Proper Form). *Given a matrix* $R \in \mathbb{C}^{n\times n}$ *that satisfies* (1.3.21) *and* (1.3.22) *for some full rank* $G \in \mathbb{C}^{n\times r}$, *start with* $G_0 = G$ *and perform the following steps for* $i = 0, \ldots, n-1$:

1. *Let* g_i *be the top row of* G_i *and let* F_i *be the submatrix obtained by deleting the leading* i *rows and columns of* F.

2. *If* $g_i J g_i > 0$, *then transform* g_i *to proper form according to* (1.7.1), *compute* l_i *and* d_i *using* (1.7.3), *and update* G_i *to* G_{i+1} *according to* (1.7.2).

3. *If* $g_i J g_i < 0$, *then transform* g_i *to proper form according to* (1.7.4), *compute* l_i *and* d_i *using* (1.7.6), *and update* G_i *to* G_{i+1} *according to* (1.7.5).

◇

1.7.4 An Associated Transmission Line

We may return to the elementary section (1.6.34) and note that its form simplifies in the proper case. Assume $g_i J g_i^* > 0$ (a similar argument holds for $g_i J g_i^* < 0$). It then follows that (1.6.27) collapses to the form

$$\begin{bmatrix} F_i l_i & G_i \end{bmatrix} \begin{bmatrix} f_i^* & \frac{\delta_i}{d_i}\begin{bmatrix} 1 & 0 \end{bmatrix} \\ \Theta_i \begin{bmatrix} \delta_i \\ 0 \end{bmatrix} & \Theta_i \begin{bmatrix} -f_i & 0 \\ 0 & I_{r-1} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} l_i & \begin{matrix} 0 \\ G_{i+1} \end{matrix} \end{bmatrix}, \tag{1.7.7}$$

which leads to the transfer matrix

$$\Theta_i(z) = \Theta_i \begin{bmatrix} -f_i & 0 \\ 0 & I_{r-1} \end{bmatrix} + \Theta_i \begin{bmatrix} \delta_i \\ 0 \end{bmatrix} (z^{-1} - f_i^*)^{-1} \frac{\delta_i}{d_i} \begin{bmatrix} 1 & 0 \end{bmatrix}$$

or, equivalently,

$$\Theta_i(z) = \Theta_i \begin{bmatrix} B_i(z) & 0 \\ 0 & I_{r-1} \end{bmatrix}, \quad B_i(z) = \frac{z - f_i}{1 - f_i^* z}. \tag{1.7.8}$$

Any $(p+q)$-row input to the above $\Theta_i(z)$ is processed by the rotation Θ_i first and then only the leading entry of the result is filtered by the all-pass function $B_i(z)$. This is represented schematically in Fig. 1.4, where the top line of the elementary section includes the factor $B_i(z)$.

When $F = Z$, the cascade of such sections is a classical discrete transmission line; it can be used to show that the generalized Schur algorithm gives a natural solution (and indicates several generalizations) of the classical inverse scattering problems (see, e.g., [BK87a], [BK87b], [CSK99]).

1.7.5 Shift-Structured ($F = Z$) Positive-Definite Matrices

Several simplifications occur when $F = Z$ and R is positive definite, say,

$$R - ZRZ^* = GJG^*, \quad R > 0. \tag{1.7.9}$$

To begin with, since the diagonal entries of Z are now zero, the expression (1.6.12) for Φ_i becomes simply $\Phi_i = Z$. (We continue to write Z, except when otherwise stated, to denote a shift matrix of any dimension.)

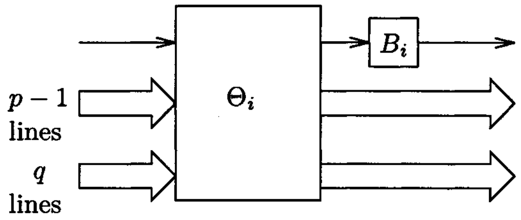

Figure 1.4. Elementary section $\Theta_i(z)$ when $g_iJg_i^* > 0$.

Moreover, since the f_i are now all zero, we conclude from (1.6.8) and from the positivity of the d_i that $g_iJg_i^* > 0$ always. This means that the proper form of the generalized Schur algorithm in this case becomes

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = G_i\,\Theta_i \begin{bmatrix} 0 & 0 \\ 0 & I_{r-1} \end{bmatrix} + Z\,G_i\,\Theta_i \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}. \tag{1.7.10}$$

Likewise, the expressions (1.6.7) and (1.6.8) for l_i and d_i become

$$l_i = \delta_i^* G_i \Theta_i \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad d_i = |\delta_i|^2. \tag{1.7.11}$$

Equation (1.7.10) yields the following simple statements:

1. Transform G_i into proper form with respect to its first column by using a J-unitary rotation Θ_i.

2. Shift down the first column and keep the rest of the columns unaltered.

1.7.6 The Classical Schur Algorithm

In order to justify our earlier claim that Alg. 1.6.2 is a far-reaching generalization of the celebrated algorithm of Schur, we focus on a special subclass of (1.7.9), viz., matrices with displacement rank 2. Our objective is to verify that in this case the generator recursion (1.7.10) collapses to Schur's original algorithm [Sch17].

For this purpose, we shall now consider semi-infinite matrices $\mathcal{R}$ with semi-infinite generator matrices $\mathcal{G}$, say,

$$\mathcal{R} - \mathcal{Z}\mathcal{R}\mathcal{Z}^* = \mathcal{G} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \mathcal{G}^*, \tag{1.7.12}$$

where we shall denote the individual entries of $\mathcal{G}$ by

$$\mathcal{G} = \begin{bmatrix} x_{00} & y_{00} \\ x_{10} & y_{10} \\ x_{20} & y_{20} \\ x_{30} & y_{30} \\ \vdots & \vdots \end{bmatrix}.$$

We further associate with the columns of $\mathcal{G}$ two power series $x_0(z)$ and $y_0(z)$:

$$\begin{aligned} x_0(z) &= x_{00} + x_{10}z + x_{20}z^2 + \cdots, \\ y_0(z) &= y_{00} + y_{10}z + y_{20}z^2 + \cdots. \end{aligned}$$

(We assume $\mathcal{R}$ and $\mathcal{G}$ are such that these power series are well defined in some region of the complex plane.) Equivalently, these power series can be obtained by multiplying $\mathcal{G}$ from the left by $\begin{bmatrix} 1 & z & z^2 & z^3 & \ldots \end{bmatrix}$,

$$\begin{bmatrix} x_0(z) & y_0(z) \end{bmatrix} = \begin{bmatrix} 1 & z & z^2 & z^3 & \ldots \end{bmatrix} \mathcal{G}.$$

Now the first step of the generator recursion (1.7.10) takes the form

$$\begin{bmatrix} 0 \\ \mathcal{G}_1 \end{bmatrix} = \mathcal{G}_0\,\Theta_0 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} + \mathcal{Z}\,\mathcal{G}_0\,\Theta_0 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \tag{1.7.13}$$

where we recall that the purpose of the J-unitary rotation Θ_0 is to annihilate y_{00}. This can be achieved by using a hyperbolic rotation of the form (cf. the discussion in App. B)

$$\Theta_0 = \frac{1}{\sqrt{1-|\gamma_0|^2}} \begin{bmatrix} 1 & -\gamma_0 \\ -\gamma_0^* & 1 \end{bmatrix}, \quad \gamma_0 = \frac{y_{00}}{x_{00}} = \frac{y_0(0)}{x_0(0)}.$$

(The positive definiteness of $\mathcal{R}$ guarantees $|x_{00}|^2 - |y_{00}|^2 > 0$ and, hence, x_{00} cannot be zero.) After applying Θ_0 and then shifting down the first column, we obtain $\mathcal{G}_1$ by

$$\begin{bmatrix} 0 \\ \mathcal{G}_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ x_{11} & y_{11} \\ x_{21} & y_{21} \\ x_{31} & y_{31} \\ \vdots & \vdots \end{bmatrix}.$$

We also associate two power series with the nonidentically zero part of the columns of $\mathcal{G}_1$,

$$\begin{aligned} x_1(z) &= x_{11} + x_{21}z + x_{31}z^2 + \cdots, \\ y_1(z) &= y_{11} + y_{21}z + y_{31}z^2 + \cdots. \end{aligned}$$

By multiplying both sides of (1.7.13) by $\begin{bmatrix} 1 & z & z^2 & z^3 & \ldots \end{bmatrix}$ from the left, and by noting that

$$\begin{bmatrix} 1 & z & z^2 & z^3 & \ldots \end{bmatrix} \mathcal{Z} = z \begin{bmatrix} 1 & z & z^2 & z^3 & \ldots \end{bmatrix},$$

we conclude that

$$\begin{bmatrix} zx_1(z) & zy_1(z) \end{bmatrix} = \begin{bmatrix} x_0(z) & y_0(z) \end{bmatrix} \Theta_0 \begin{bmatrix} z & 0 \\ 0 & 1 \end{bmatrix}.$$

This is a functional recursion that tells us how the power series of the successive generator matrices are related to each other.

The recursive procedure now continues as follows: compute γ_1 as the ratio of y_{11} and x_{11}, multiply the prearray $\mathcal{G}_1$ by Θ_1 in order to introduce a zero in the first entry of the second column of the postarray, shift down the first column of the postarray, and so on. In function form, for the ith step, we have

$$z\begin{bmatrix} x_{i+1}(z) & y_{i+1}(z) \end{bmatrix} = \begin{bmatrix} x_i(z) & y_i(z) \end{bmatrix} \Theta_i \begin{bmatrix} z & 0 \\ 0 & 1 \end{bmatrix}, \tag{1.7.14}$$

where Θ_i is an elementary hyperbolic rotation determined by a coefficient γ_i,

$$\Theta_i = \frac{1}{\sqrt{1-|\gamma_i|^2}} \begin{bmatrix} 1 & -\gamma_i \\ -\gamma_i^* & 1 \end{bmatrix}, \quad \gamma_i = \frac{y_i(0)}{x_i(0)}. \tag{1.7.15}$$

If we introduce the (so-called scattering) function

$$s_i(z) \triangleq \frac{y_i(z)}{x_i(z)},$$

it then follows easily from (1.7.14) that $s_i(z)$ satisfies the recursion:

$$s_{i+1}(z) = \frac{1}{z} \frac{s_i(z) - \gamma_i}{1 - \gamma_i^* s_i(z)}, \quad \gamma_i = s_i(0). \tag{1.7.16}$$

This is the famous Schur recursion, which was originally derived for checking when power series are analytic and bounded by unity in the unit disc [Sch17]. We see that it follows as a special case of our earlier Alg. 1.6.2, which is in this sense a generalization of Schur's earlier work. Note that the generalization is in several respects: it allows for displacement ranks larger than 2, it allows for general lower triangular matrices F instead of the shift matrix Z, and it allows for strongly regular matrices R instead of positive-definite matrices R. The generalization also extends to non-Hermitian matrices and to other definitions of matrix structure (see further ahead and also [KS95a]). (The generating function language can continue to be used for certain forms of F—see [Lev97].)

1.7.7 The Special Case of Toeplitz Matrices

When the matrix R in (1.7.12) is a finite Hermitian Toeplitz matrix, $T = [c_{i-j}]_{i,j=0}^{n-1}$, with $c_0 = 1$, then it is easy to check that a generator matrix is given by

$$G^T = \begin{bmatrix} 1 & c_1 & \dots & c_{n-1} \\ 0 & c_1 & \dots & c_{n-1} \end{bmatrix}, \quad J = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

(see (1.2.5)).

It turns out that in this case, the so-called Schur coefficients γ_i coincide with the reflection coefficients introduced in the Levinson–Durbin algorithm (1.2.7)–(1.2.9) for solving the Yule–Walker equations, which is why we used the same symbols for them. But, as noted earlier (at the end of Sec. 1.2), we can now present an alternative method of solving the Yule–Walker equations—we compute the $\{\gamma_i\}$ by the Schur algorithm and then directly use the recursions (1.2.7), thus circumventing the inner-product evaluations (1.2.8) of the Levinson–Durbin algorithm. This is sometimes called the hybrid method of solving the Yule–Walker equations—the algorithm can also be extended to solving Toeplitz linear equations with arbitrary right-hand sides, as will be shown in Sec. 1.8.4 below.

1.8 SOME APPLICATIONS IN MATRIX COMPUTATION

In the previous sections, we established that displacement structure is *invariant* under Schur complementation and derived a fast algorithm for recursively updating the generator matrices of the successive Schur complements of a structured matrix.

We also mentioned earlier that the above invariance property is very useful in handling combinations of structured matrices. Now that we have derived the fast algorithm, we can return to this earlier remark and demonstrate its usefulness by presenting a few examples. The examples we present here will also highlight the value of extending the the definition of displacement structure by replacing Z in (1.2.4) by more general (say, strictly lower) triangular matrices F in (1.3.21). Such extension allows us to handle, through a technique known as *embedding* [CKL87], [KC94], many matrix computation problems that involve combinations of structured matrices.

1.8.1 Going Beyond $F = Z$

A first application of the embedding idea is to note that T^{-1} is the Schur complement of the $(1,1)$ block in the Toeplitz block matrix

$$M = \begin{bmatrix} -T & I \\ I & 0 \end{bmatrix}. \tag{1.8.1}$$

Now by examining $M - Z_{2n}MZ_{2n}$, we can see that the displacement rank of M is less than or equal to 4, where we have employed the notation Z_{2n} to denote the $2n \times 2n$ lower shift matrix. Therefore, by Lemma 1.5.2, the rank of the Schur complement, T^{-1}, must also be less than or equal to 4. However, this is a weak conclusion, because we know from Lemma 1.5.1 that the displacement rank of T^{-1} is 2.

If we instead employ the definition

$$\nabla M = M - \begin{bmatrix} Z_n & 0 \\ 0 & Z_n \end{bmatrix} M \begin{bmatrix} Z_n & 0 \\ 0 & Z_n \end{bmatrix}^*, \tag{1.8.2}$$

where we use $F = (Z_n \oplus Z_n)$ in the definition $R - FRF^*$ rather than $F = Z_{2n}$, then it is easy to see that the displacement rank of M is now 2. In fact, the reader may wish to check that for a Hermitian Toeplitz matrix $T = [c_{i-j}]_{i,j=0}^{n-1}$, $c_0 = 1$, we obtain

$$M - FMF^* = GJG^*, \quad F = Z_n \oplus Z_n,$$

where $J = (1 \oplus -1)$ and

$$G^{\mathrm{T}} = \begin{bmatrix} 0 & c_1 & \dots & c_{n-1} & 1 & 0 & \dots & 0 \\ 1 & c_1 & \dots & c_{n-1} & 1 & 0 & \dots & 0 \end{bmatrix}^{\mathrm{T}}. \tag{1.8.3}$$

Soon we shall give more elaborate examples. However, from the embedding (1.8.1), we shall show how we can obtain several interesting results on both T and T^{-1}.

1.8.2 Simultaneous Factorization of T and T^{-1}

We shall see that by applying the generalized Schur algorithm to the matrix M in (1.8.1) we can not only determine (the generators of) T^{-1} but also simultaneously factor both T and T^{-1}. Thus consider the situation after we apply n steps of the generalized Schur algorithm (say Alg. 1.6.2 or, in proper form, Alg. 1.7.1) to the generator G of M, viz., (1.8.3). Of course, we shall then get a generator, say, $\{a, b\}$, of the Schur complement T^{-1} from which we can recover the matrix T^{-1} as

$$T^{-1} = \mathcal{L}(a)\mathcal{L}^*(a) - \mathcal{L}(b)\mathcal{L}^*(b).$$

Suppose that we also (or only) want the triangular factors of T^{-1}. One way to get these is to apply the Schur algorithm to the generator $\{a, b\}$. But in fact the factors of

T^{-1} are already available from the results of the first n steps of the generalized Schur algorithm applied to the generator of M.

To clarify this, assume we apply the first n recursive steps of the generalized Schur algorithm to a generator of the $2n \times 2n$ matrix M, with $F = (Z_n \oplus Z_n)$. This provides us with the first n columns and the first n diagonal entries of the triangular factors of M, which we denote by L_{2n} and D_{2n}. That is, we obtain the first n columns of L_{2n} and the first n entries of D_{2n} in the factorization $M = L_{2n}D_{2n}^{-1}L_{2n}^*$. Let us denote the leading $n \times n$ block of D_{2n} by D and let us partition the first n columns of L_{2n} into the form

$$\begin{bmatrix} L \\ U \end{bmatrix},$$

where L is $n \times n$ lower triangular and U is an $n \times n$ matrix that we shall soon see has to be upper triangular. It follows from the Schur reduction interpretation of Alg. 1.6.1 that we must have

$$\begin{bmatrix} -T & I \\ I & 0 \end{bmatrix} = M = \begin{bmatrix} L \\ U \end{bmatrix} D^{-1} \begin{bmatrix} L^* & U^* \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & T^{-1} \end{bmatrix}.$$

By equating terms on both sides of the above equality we conclude that $U = L^{-*}D$, $-T^{-1} = UD^{-1}U^*$, and $-T = LD^{-1}L^*$. Hence, the first n recursive steps of the algorithm provide not only the triangular factorization of T but also the triangular factorization of T^{-1}. In fact, by examining the form (1.8.3) of the generator for the matrix M, the reader can check that the generalized Schur algorithm for the present problem (i.e., for M as in (1.8.1)) is exactly what we called a hybrid algorithm in Sec. 1.7.7.

Unfortunately, this nice embedding idea does not extend to a general F. To see this, consider again the simple example of a Pick matrix P,

$$P = \left[\frac{u_i u_j^* - v_j v_j^*}{1 - f_i f_j^*} \right]_{i,j=0}^{n-1}, \tag{1.8.4}$$

where the $\{u_i, v_i\}$ are row vectors of dimensions p and q, respectively, and the f_i are complex points inside the open unit disc ($|f_i| < 1$). Then with $F = \text{diag}\{f_0, f_1, \ldots, f_{n-1}\}$, we can check that P has ∇_F-displacement

$$P - FPF^* = \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \end{bmatrix} \begin{bmatrix} I_p & 0 \\ 0 & -I_q \end{bmatrix} \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \end{bmatrix}^*. \tag{1.8.5}$$

However, if we now construct the matrix M as before, i.e., as in (1.8.1) but with P instead of T, it is easy to check that $M - (F \oplus F)M(F \oplus F)^*$ does not have low rank since $(I - FF^*)$ is full rank in general.

A general method that overcomes this difficulty has recently been obtained and will be described in Sec. 1.10, where this example will be reconsidered.

1.8.3 QR Factorization of Structured Matrices

We now show how the displacement ideas can be used to suggest a fast algorithm for the QR factorization of structured matrices (with Q unitary and R upper triangular). This has been a much-studied problem, starting with the dissertation [Swe82]. Some later papers are those of [BBH86], [Cyb83], [Cyb87], [Swe84]. The displacement approach described below is much simpler, conceptually and algebraically.

Let X be an $n \times n$ matrix[4] that has low displacement rank with respect to the displacement $X - Z_n X Z_n^*$. Form the displacement of

$$M = \begin{bmatrix} -I & X & 0 \\ X^* & 0 & X^* \\ 0 & X & 0 \end{bmatrix}$$

with $F = Z_n \oplus Z_n \oplus Z_n$ and find a generator for M. A general procedure for doing this has been given in [Chu89], [KC94]; in many cases, e.g., when X is Toeplitz, one can obtain a generator of length 5 almost by inspection (see, e.g., the discussion in Sec. 3.4 of this book).

After n steps of the generalized Schur algorithm applied to a generator of M, we shall have a generator of

$$M_1 = \begin{bmatrix} X^*X & X^* \\ X & 0 \end{bmatrix}.$$

After another n steps, we shall have the partial triangularization (where L is $n \times n$ lower triangular and U is an $n \times n$ matrix)

$$M_1 = \begin{bmatrix} L \\ U \end{bmatrix} D^{-1} \begin{bmatrix} L^* & U^* \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -I \end{bmatrix}.$$

By equating terms on both sides of the above equality we conclude that

$$X^*X = (LD^{-*/2})(LD^{-*/2})^* \ , \quad (UD^{-*/2})(LD^{-*/2})^* = X,$$

and $(UD^{-*/2})(UD^{-*/2})^* = I$. Therefore, we can identify

$$Q = UD^{-*/2}, \qquad R = (LD^{-*/2})^*$$

as the Q and R factors in the QR factorization of X. Here, D is a positive-definite diagonal matrix and $D^{1/2}$ denotes a diagonal matrix whose entries are the square roots of the diagonal entries of D. In summary, the QR factors of the structured matrix X can be obtained by applying the Schur recursion to a properly defined extended structured matrix M.

The above procedure may encounter numerical difficulties in finite precision implementations. However, in Sec. 3.1, it will be shown how to employ such embedding constructions to develop the first provably backward-stable algorithm for the solution of linear systems of equations with structured coefficient matrices (cf. [CS98]).

1.8.4 Avoiding Back Substitution in Linear Equations

The previous examples all involved the Schur algorithm for Hermitian matrices. Here is an example of a problem involving non-Hermitian matrices. The generalized Schur algorithm for non-Hermitian matrices is similar in nature to what we described in Sec. 1.6 and is covered in detail in [KS95a] (and briefly in Sec. 1.11). It is not necessary to know the exact algorithm to follow the present discussion.

[4]The argument also applies to rectangular matrices, say, $m \times n$.

Consider a linear system of equations of the form

$$Tx = b,$$

where T is an $n \times n$ strongly regular Hermitian Toeplitz matrix and b is a known column vector. One possibility for determining the entries of x is the following: compute the triangular factorization of T, say,

$$T = LD^{-1}L^*,$$

and then solve, via back substitution, the triangular system of equations in y and x,

$$LD^{-1}y = b \quad \text{and} \quad L^*x = y.$$

A major drawback of a back-substitution step is that it involves serial operations and does not lend itself to a parallelizable algorithm.

A way out of this is to employ a bordering (or embedding) technique (see, e.g., [KC94]). For this purpose, we define the extended (non-Hermitian) matrix

$$R = \begin{bmatrix} -T & b \\ I & 0 \end{bmatrix}$$

and note that the Schur complement of $-T$ in R is precisely $T^{-1}b$, which is equal to the desired solution x. Now the matrix R itself is also structured since T is Toeplitz. More precisely, we know that $T - ZTZ^*$ has rank 2 and it follows that

$$\begin{bmatrix} -T & b \\ I & 0 \end{bmatrix} - \begin{bmatrix} Z & \\ & Z \end{bmatrix} \begin{bmatrix} -T & b \\ I & 0 \end{bmatrix} \begin{bmatrix} Z & \\ & 0 \end{bmatrix}^* \quad \text{also has low rank.}$$

Therefore, after n steps of partial triangularization of R, we shall have a generator of its Schur complement, from which we can read out the solution x.

There are several other interesting examples and applications (see, e.g., [CXT94], [AS99]), but let us move on.

1.9 LOOK-AHEAD (BLOCK) SCHUR ALGORITHM

A standing assumption in all the preceding has been that the structured matrix R is strongly regular, i.e., all its leading minors are nonzero. However there are applications where we may have some poorly conditioned or even zero leading minors. In such cases, one can use the smallest nonsingular leading minor, or a well-conditioned leading minor of appropriate dimensions, in order to proceed with a block Schur complementation step. The use of such block pivoting has been studied by several authors trying to devise effective numerical algorithms for various classes of structured matrices (see, e.g., [CH92b], [Gut93], [Fre94], and the references therein).

There are also several theoretically interesting studies on the special case of Hankel matrices with zero minors (for which there is the celebrated Berlekamp–Massey algorithm [Mas69] and related studies of the so-called partial realization problem of system theory) and the less-studied problem for Toeplitz matrices (see [Pal90], [PK93], and the references therein).

Here we shall describe a very general algorithm that goes considerably beyond the results noted so far. The results appeared first in [SK95b], [Say92]; here we present an array-based derivation.

Consider a Hermitian and invertible (but not necessarily strongly regular) matrix $R \in \mathbb{C}^{n\times n}$, and let η_0 denote the desired size of the leading invertible block, D_0, with respect to which a Schur complementation step is to be performed. The η_0 may stand for the size of the smallest nonsingular minor of R or, alternatively, for the size of a numerically well-conditioned block. If L_0 represents the first η_0 columns of R, then we can replace the earlier Schur complementation step (1.6.4) by the block step

$$R - L_0 D_0^{-1} L_0^* = \begin{bmatrix} 0_{\eta_0\times\eta_0} & 0 \\ 0 & R_1 \end{bmatrix},$$

where R_1 is now an $(n-\eta_0)\times(n-\eta_0)$ matrix that is the Schur complement of D_0 in R. We are also being explicit about the dimensions of the leading zero block in the resulting matrix, viz., $\eta_0 \times \eta_0$.

The matrix L_0 is $n\times\eta_0$ with a leading $\eta_0\times\eta_0$ block that is equal to D_0. If we further let η_1 denote the desired size of the leading invertible block of R_1 (denoted by D_1) and consider the corresponding first η_1 columns of R_1 (denoted by L_1), then we write for our second block step

$$R_1 - L_1 D_1^{-1} L_1^* = \begin{bmatrix} 0_{\eta_1\times\eta_1} & 0 \\ 0 & R_2 \end{bmatrix},$$

where R_2 is an $(n-\eta_0-\eta_1)\times(n-\eta_0-\eta_1)$ matrix that is the Schur complement of D_1 in R_1. Repeating this block Schur reduction procedure, viz.,

$$\begin{bmatrix} 0_{\eta_i\times\eta_i} & 0 \\ 0 & R_{i+1} \end{bmatrix} = R_i - L_i D_i^{-1} L_i^*, \quad i \geq 0, \tag{1.9.1}$$

we clearly get, after, say, t steps,

$$R = LD^{-1}L^*$$
$$= L_0 D_0^{-1} L_0^* + \begin{bmatrix} 0_{\eta_0\times\eta_1} \\ L_1 \end{bmatrix} D_1^{-1} \begin{bmatrix} 0_{\eta_0\times\eta_1} \\ L_1 \end{bmatrix}^* + \begin{bmatrix} 0_{\eta_0\times\eta_2} \\ 0_{\eta_1\times\eta_2} \\ L_2 \end{bmatrix} D_2^{-1} \begin{bmatrix} 0_{\eta_0\times\eta_2} \\ 0_{\eta_1\times\eta_2} \\ L_2 \end{bmatrix}^* + \cdots,$$

where $D = (D_0 \oplus D_1 \oplus \ldots \oplus D_{t-1})$ is now *block* diagonal and the (nonzero parts of the) columns of the *block* lower triangular matrix L are $\{L_0,\ldots,L_{t-1}\}$. Here t is the number of reduction steps and, hence, $n = \sum_{i=0}^{t-1}\eta_i$.

The computational cost of this block reduction procedure is, as mentioned earlier, $O(n^3)$. By exploiting the structure of R we can reduce the computational cost by deriving an algorithm that operates on the successive generator matrices instead. So assume R satisfies (1.3.21) and (1.3.22). It is then clear that the same array equation (1.6.15) still holds, viz.,

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix}\Omega = \begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix}, \quad \Omega(D^{-1}\oplus J)\Omega^* = (D^{-1}\oplus J), \tag{1.9.2}$$

except that D and L are now *block* diagonal and *block* lower triangular, and U is such that $U = L^{-*}D$.

The same recursive argument that we employed in Sec. 1.6.4 shows that the basic recursion for the generators of the successive (block) Schur complements R_i now takes the form

$$\left[\begin{array}{c|c} F_i L_i & G_i \end{array}\right]\Sigma_i = \left[\begin{array}{c|c} L_i & \begin{matrix} 0 \\ G_{i+1} \end{matrix} \end{array}\right], \tag{1.9.3}$$

where L_i satisfies the equation

$$L_i - F_i L_i \hat{F}_i^* = G_i J \hat{G}_i^*, \tag{1.9.4}$$

and $\hat{F}_i$ is the leading $\eta_i \times \eta_i$ block of F_i, which is now partitioned as

$$F_i = \begin{bmatrix} \hat{F}_i & 0 \\ \star & F_{i+1} \end{bmatrix}.$$

That is, F_{i+1} is now obtained by deleting the leading η_i rows and columns of F_i. Likewise, $\hat{G}_i$ denotes the top η_i rows of G_i and Σ_i is $(D_i^{-1} \oplus J)$-unitary. The quantities $\{\hat{F}_i, \hat{G}_i, L_i, D_i\}$ play the role of the quantities $\{f_i, g_i, l_i, d_i\}$ that we encountered earlier in Sec. 1.6.4.

If we denote the entries of Σ_i by

$$\Sigma_i = \begin{bmatrix} A_i & B_i \\ C_i & S_i \end{bmatrix}, \tag{1.9.5}$$

where A_i is $\eta_i \times \eta_i$, B_i is $\eta_i \times r$, C_i is $r \times \eta_i$, and S_i is $r \times r$, we can verify that we must have

$$A_i = \hat{F}_i^*, \qquad C_i = J\hat{G}_i^*. \tag{1.9.6}$$

To identify the remaining entries $\{B_i, S_i\}$, we note that in view of the $(D_i^{-1} \oplus J)$-unitarity of Σ_i, the entries $\{A_i, B_i, C_i, S_i\}$ must satisfy

$$\begin{bmatrix} A_i & B_i \\ C_i & S_i \end{bmatrix} \begin{bmatrix} D_i^{-1} & 0 \\ 0 & J \end{bmatrix} \begin{bmatrix} A_i & B_i \\ C_i & S_i \end{bmatrix}^* = \begin{bmatrix} D_i^{-1} & 0 \\ 0 & J \end{bmatrix}. \tag{1.9.7}$$

Following the derivation in the proof of Lemma 1.6.4, we can use this relation to identify $\{B_i, S_i\}$ and obtain the following algorithm [Say92], [SK95b].

Algorithm 1.9.1 (Block or Look-Ahead Schur Algorithm). *Given a matrix $R \in \mathbb{C}^{n\times n}$ that satisfies* (1.3.21) *and* (1.3.22) *for some full rank $G \in \mathbb{C}^{n\times r}$, start with $F_0 = F$, $G_0 = G$ and perform the following steps:*

1. *At step i we have F_i and G_i. Let $\hat{G}_i$ denote the top η_i rows of G_i and let $\hat{F}_i$ denote the leading $\eta_i \times \eta_i$ block of F_i.*

2. *The ith triangular factors L_i and D_i are the solutions of the equations*

$$D_i = \hat{F}_i D_i \hat{F}_i^* + \hat{G}_i J \hat{G}_i^*, \quad L_i = F_i L_i \hat{F}_i^* + G_i J \hat{G}_i^*. \tag{1.9.8}$$

3. *Update the generator matrix G_i as follows:*

$$\begin{bmatrix} 0_{\eta_i \times r} \\ G_{i+1} \end{bmatrix} = \left\{ G_i + (\tau_i^* F_i - I_{n-\alpha_i}) L_i D_i^{-1} (I_{\eta_i} - \tau_i^* \hat{F}_i)^{-1} \hat{G}_i \right\} \Theta_i,$$

where Θ_i is an arbitrary J-unitary matrix and τ_i is an arbitrary unit-modulus scalar. Also, $\alpha_i = \sum_{k=0}^{i-1} \eta_k$.

The matrix G_i is a generator for R_i,

$$R_i - F_i R_i F_i^* = G_i J G_i^*,$$

which is the Schur complement of R with respect to its leading $\alpha_i \times \alpha_i$ block.

◇

We may remark that although block algorithms have often been used in connection with poorly conditioned matrices, Alg. 1.9.1 is quite general and has several other applications.

1.10 FAST INVERSION OF STRUCTURED MATRICES

Our discussion so far has been mainly concerned with the direct factorization problem, viz., that of computing the triangular factors of R. In Sec. 1.8.2 we saw how the embedding technique allowed us to employ the fast Schur algorithm for factoring the inverse of a Toeplitz matrix as well. We remarked, however, at the end of Sec. 1.8.2 that this technique does not extend readily to more general structured matrices. In fact, similar results for computing the triangular factors of the *inverse* matrix R^{-1} have not yet been obtained for the general case.

For Toeplitz matrices, as we mentioned in Sec. 1.2, the first and best-known algorithm for factoring the inverse is the celebrated Levinson–Durbin algorithm. Early attempts at extending this algorithm beyond the Toeplitz case were made in [FKML78], [Del82], [DGK85], [Lev83] but the formulas were rather complicated. After the (re)discovery of the Schur algorithm for directly factoring Toeplitz matrices rather than their inverses, it was realized by several authors (see [KH83], [Kai85]) that the Levinson–Durbin algorithm could be replaced by a two-step procedure: use the Schur algorithm to compute the so-called reflection coefficients, and then use a simple recursion to compute the triangular factors of the inverse; this is actually the hybrid algorithm of Sec. 1.8.2. This two-step procedure requires slightly more computation than the Levinson–Durbin algorithm on a serial machine, but it is significantly less expensive on a parallel machine. (This is because the classical Levinson–Durbin algorithm obtains the reflection coefficients via certain inner products, which cannot be parallelized in an efficient manner.) The hybrid method was extended in [Chu89] and [KC94] to invert matrices obeying displacement equations of the form $R - FRF^* = GJG^*$, where F had some *strictly* lower triangular structure. However, these extended algorithms generally require an intermediate array whose displacement rank can be larger than r.

In the recent works [BSLK96], [BKLS98a], [BKLS98b] we removed all the above-mentioned restrictions. In this section we provide an overview of the solution method. However, for simplicity of presentation, here we shall first assume that the following *nondegeneracy* condition holds (in addition to (1.3.22)):

$$i \neq j \quad \Longrightarrow \quad f_i \neq f_j\,, \qquad i,j \in \{0,1,\ldots,n-1\}\,. \tag{1.10.1}$$

This condition simplifies the derivation of the recursions for the factorization of the inverse of R; it is not needed for the direct factorization problem itself as we saw in the earlier section. The assumption, however, excludes the important cases of $F = Z$ or F strictly lower triangular; the general case is briefly discussed in Sec. 1.10.7.

1.10.1 Schur Construction (Inflation)

The Schur reduction procedure of Alg. 1.6.1 can be extended to the factorization of the inverse matrix. While we factored R before by recursively deflating it, we now factor R^{-1} by "inflation."

Recall that we expressed the triangular factorizations of R and R^{-1} in the somewhat nontraditional forms (cf. (1.6.1) and (1.6.14))

$$R = L\,D^{-1}\,L^*, \qquad R^{-1} = U\,D^{-1}\,U^*, \tag{1.10.2}$$

where $D = \text{diag}\{\,d_0,\ d_1,\ \ldots,\ d_{n-1}\}$ and

$$LD^{-1}U^* = I_n\,. \tag{1.10.3}$$

The nonzero parts of the columns of L are denoted by $\{l_j\}$ and of U are denoted by $\{u_j\}$.

Algorithm 1.10.1 (Schur Construction for Inversion). *Given $R \in \mathbb{C}^{n\times n}$, start with $\Delta_0 = [\ ]$ (the empty matrix) and repeat the following steps for $i = 0, 1, \ldots, n-1$:*

1. *Let l_i and d_i denote the first column and the upper-left-corner element of R_i. These can be evaluated via the Schur reduction of Alg.* 1.6.1.

2. *Given $\{l_0, l_1, \ldots, l_i\}$ and $\{d_0, d_1, \ldots, d_i\}$, compute u_i from the equation*

$$\sum_{j=0}^{i} \check{l}_j d_j^{-1} \check{u}_j^* = \begin{bmatrix} I_{i+1} & 0 \\ 0 & 0 \end{bmatrix},$$

where

$$\check{l}_j = \begin{bmatrix} 0 \\ l_j \end{bmatrix} \begin{matrix} \}j \\ \}n-j \end{matrix}, \qquad \check{u}_j = \begin{bmatrix} u_j \\ 0 \end{bmatrix} \begin{matrix} \}j+1 \\ \}n-j-1 \end{matrix}.$$

3. *Perform the Schur construction step*

$$\Delta_{i+1} = \begin{bmatrix} \Delta_i & 0 \\ 0 & 0 \end{bmatrix} + u_i d_i^{-1} u_i^*. \tag{1.10.4}$$

◇

Observe that while Alg. 1.6.1 involves *reduction* steps, i.e., rank 1 matrices are recursively subtracted at each step, the above algorithm involves *construction* steps in which rank 1 matrices are added to Δ_i (after bordering the matrix with zeros). Thus this procedure successively constructs the rows and columns of R^{-1}. The intermediate array Δ_i is the Schur complement of the trailing $(n-i) \times (n-i)$ block in R^{-1}. At the end of the procedure we obtain $\Delta_{n-1} = R^{-1}$. It is also immediate to verify the following result.

Lemma 1.10.1 (Partitioning of the Triangular Factorization). *If we partition R^{-1} and U as*

$$R^{-1} = \begin{array}{c} \begin{matrix} \overbrace{}^{i} & \overbrace{}^{n-i} \end{matrix} \\ \begin{bmatrix} W_i & V_i^* \\ V_i & T_i \end{bmatrix} \end{array} \begin{matrix} \}i \\ \}n-i \end{matrix}, \qquad U = \begin{array}{c} \begin{matrix} \overbrace{}^{i} & \overbrace{}^{n-i} \end{matrix} \\ \begin{bmatrix} \bar{U}_i & \tilde{U}_i \\ 0 & U_i \end{bmatrix} \end{array} \begin{matrix} \}i \\ \}n-i \end{matrix}$$

and let D be partitioned as before (after Alg. 1.6.1*), then the trailing principal block T_i and its Schur complement $\Delta_i = W_i - V_i^* T_i^{-1} V_i$ admit the following triangular decompositions:*

$$T_i = U_i D_i^{-1} U_i^*, \qquad \Delta_i = \bar{U}_i \bar{D}_i^{-1} \bar{U}_i^*.$$

Moreover, we also have that $R_i^{-1} = T_i$ and $\Delta_i^{-1} = P_i$.

◇

In other words, the inverse of the trailing principal block in R^{-1} is equal to the Schur complement of the leading principal block in R and vice versa: the inverse of the leading principal block in R is equal to the Schur complement of the trailing principal block in R^{-1}.

1.10.2 Statement of the Fast Inversion Algorithm

Just as for R_i, it turns out that Δ_i inherits the displacement structure of R, as we shall now proceed to show. Using this property, we shall also be able to reduce the cost of the above Schur reduction procedure from $O(n^3)$ to $O(n^2)$.

The algorithm we state below has one very important feature: it does not require that we know a priori a generator matrix H^* for R^{-1} to factor R^{-1}. Instead, it works directly with the given displacement description of R itself, namely, $\{F, G, J\}$, and constructs $\{H^*, U\}$! This will be achieved by recursively computing matrices $H_i \in \mathbb{C}^{i\times i}$ $(i=0,1,\ldots,n-1)$ that are generators for the successive Δ_i,

$$\Delta_i - \bar{F}_i^* \Delta_i \bar{F}_i = H_i^* J H_i, \tag{1.10.5}$$

where, according to (1.6.6), $\bar{F}_i$ is the leading $i \times i$ block of F,

$$F = \begin{array}{c} \begin{array}{cc} \overbrace{\phantom{\bar{F}_i}}^{i} & \overbrace{}^{n-i} \end{array} \\ \begin{bmatrix} \bar{F}_i & 0 \\ \tilde{F}_i & F_i \end{bmatrix} \end{array} \begin{array}{l} \\ \}\, i \\ \}\, n-i \end{array} .$$

At this point, we encourage the reader to review Alg. 1.6.2 for the factorization of R, because it will be used here.

Algorithm 1.10.2 (Schur Algorithm for the Inverse Matrix). *Given a matrix $R \in \mathbb{C}^{n\times n}$ that satisfies* (1.3.21), (1.3.22), *and* (1.10.1) *for some full rank $G \in \mathbb{C}^{n\times r}$, start with $H_0 = [\ ]$ and repeat the following steps for $i=0,\ldots,n-1$:*

1. *Let $\{l_i, d_i, g_i, G_i\}$ be defined and computed as in Alg.* 1.6.2.

2. *Let $\bar{F}_{i+1}$ and u_i be partitioned as*

$$\bar{F}_{i+1} = \begin{bmatrix} \bar{F}_i & 0 \\ \varphi_i & f_i \end{bmatrix}, \qquad u_i = \begin{bmatrix} \bar{u}_i \\ 1 \end{bmatrix}. \tag{1.10.6}$$

Compute $\bar{u}_i$ by solving the linear system

$$(\bar{F}_i^* - f_i^* I_i)\,\bar{u}_i = H_i^* J g_i^* - \varphi_i^*. \tag{1.10.7}$$

The nondegeneracy condition (1.10.1) *ensures that* (1.10.7) *has a unique solution for $\bar{u}_i$.*

3. *Apply the same transformation Σ_i as in* (1.6.9) *to the prearray shown below and obtain H_{i+1}^*:*

$$\begin{bmatrix} u_i & \begin{matrix} H_i^* \\ 0 \end{matrix} \end{bmatrix} \Sigma_i = \begin{bmatrix} \bar{F}_{i+1}^* u_i & H_{i+1}^* \end{bmatrix}. \tag{1.10.8}$$

Notice that H_{i+1}^* has one more row than H_i^*.

◇

In fact, we can further show that H_{i+1}^* can be obtained from H_i^* explicitly via the equation

$$H_{i+1}^* = \left\{ \begin{bmatrix} H_i^* \\ 0 \end{bmatrix} + \begin{bmatrix} \Psi_i - I_i & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} H_i^* \\ 0 \end{bmatrix} \frac{J g_i^* g_i}{g_i J g_i^*} + \begin{bmatrix} \phi_i \\ 1 \end{bmatrix} g_i d_i^{-1} \right\} \Theta_i ,$$

where Θ_i is any J-unitary matrix Θ_i and

$$\Psi_i = (\bar{F}_i^* - f_i^* I_i)^{-1}(I_i - f_i \bar{F}_i^*), \tag{1.10.9}$$

$$\phi_i = -(\bar{F}_i^* - f_i^* I_i)^{-1}\varphi_i^*. \tag{1.10.10}$$

1.10.3 Algorithm Development

The derivation of the algorithm follows from the array-based arguments that we employed earlier in Sec. 1.6.4. Recall that in that section we focused on the top block row of both the pre- and postarrays in (1.6.15). By further considering the effect of the rotation Ω on the second block row of both arrays, we are led to the above algorithm.

More explicitly, we already know from the argument that led to the array equation (1.6.15) that there exists a $(D^{-1} \oplus J)$-unitary matrix Ω such that

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix} \Omega = \begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix}. \tag{1.10.11}$$

We used this equation earlier to argue recursively that by triangularizing the prearray

$$\begin{bmatrix} FL & G \end{bmatrix}$$

through a sequence of $(D^{-1} \oplus J)$-unitary rotations $\{\Omega_0, \Omega_1, \ldots, \Omega_{n-1}\}$ we obtain the generalized Schur procedure listed in Alg. 1.6.2. Now by applying these same rotations to the second block row in (1.10.11), viz.,

$$\begin{bmatrix} U & 0 \end{bmatrix},$$

we can justify Alg. 1.10.2.

For this purpose, we start by partitioning the pre- and postarrays as

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix} = \left[\begin{array}{cc|c} \bar{F}_i\bar{L}_i & 0 & \multirow{2}{*}{G} \\ \tilde{F}_i\bar{L}_i + F_i\tilde{L}_i & F_i L_i & \\ \hline \bar{U}_i & \tilde{U}_i & \multirow{2}{*}{0} \\ 0 & U_i & \end{array}\right] \tag{1.10.12}$$

and

$$\begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix} = \left[\begin{array}{cc|c} \bar{L}_i & 0 & \multirow{2}{*}{0} \\ \tilde{L}_i & L_i & \\ \hline \bar{F}_i^*\bar{U}_i & \bar{F}_i^*\tilde{U}_i + \tilde{F}_i^*U_i & \multirow{2}{*}{H^*} \\ 0 & F_i^*\, U_i & \end{array}\right]. \tag{1.10.13}$$

After i iterations, the first i columns of the postarray are already computed, while the last $(n-i)$ columns of the prearray have not yet been modified. Therefore, the ith intermediate array must have the following form:

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix} \Omega_0\Omega_1 \ldots \Omega_{i-1} = \left[\begin{array}{cc|c} \bar{L}_i & 0 & 0 \\ \tilde{L}_i & F_i\, L_i & G_i \\ \hline \bar{F}_i^*\bar{U}_i & \tilde{U}_i & H_i^* \\ 0 & U_i & 0 \end{array}\right], \tag{1.10.14}$$

where G_i and H_i^* denote the nontrivial elements that appear in the upper- and lower-right-corner blocks. Note that the prearray (1.6.22), the intermediate array (1.6.24), and the postarray (1.6.23) are all $(D^{-1} \oplus J)$-equivalent; i.e., their "squares" in the $(D^{-1} \oplus J)$ metric must be equal.

We already know from Lemma 1.6.3 that G_i is a generator matrix for the leading Schur complement R_i. Now, a similar conclusion follows for H_i^*. Indeed, note first that the matrix

$$\begin{bmatrix} G_i \\ H_i^* \end{bmatrix} \tag{1.10.15}$$

is $n \times r$ and must have full rank r. This is because the prearray in (1.6.24) has full rank $n+r$. Now since each of the Ω_j is invertible, we conclude that the postarray in (1.6.24) must also have full rank $n+r$. It then follows that (1.10.15) has full rank r.

Moreover, it also follows that the Schur complement Δ_i satisfies the displacement equation (1.10.5). This is obtained by comparing the *prearray* and the *intermediate* array. The "squared lengths" of the third block rows of (1.6.22) and (1.6.24) must be equal, i.e.,

$$\begin{aligned}&\left[\bar{U}_i \ \ \tilde{U}_i \ \ 0\right](\bar{D}_i^{-1} \oplus D_i^{-1} \oplus J)\left[\bar{U}_i \ \ \tilde{U}_i \ \ 0\right]^* \\ &= \left[\bar{F}_i^*\bar{U}_i \ \ \tilde{U}_i \ \ H_i^*\right](\bar{D}_i^{-1} \oplus D_i^{-1} \oplus J)\left[\bar{F}_i^*\bar{U}_i \ \ \tilde{U}_i \ \ H_i^*\right]^*.\end{aligned}$$

Therefore,

$$(\bar{U}_i\,\bar{D}_i^{-1}\,\bar{U}_i^*) = \bar{F}_i^*\,(\bar{U}_i\,\bar{D}_i^{-1}\,\bar{U}_i^*)\,\bar{F}_i + H_i^*\,J\,H_i\,.$$

But since $\Delta_i = \bar{U}_i\bar{D}_i^{-1}\bar{U}_i^*$, we conclude that (1.10.5) holds.

We now establish (1.10.7) and the generator recursion of Alg. 1.10.2. Recall that we used (1.6.25) earlier to derive the closed-form expression (1.6.26) for l_i. Unfortunately, a similar argument using (1.10.5) cannot be used to determine u_i. This is because Δ_i involves $u_0, \ldots, u_{i-1}$ but not u_i. However, the $(D^{-1} \oplus J)$-unitary equivalence of the *intermediate* array and the *postarray* shows that the $(D^{-1} \oplus J)$-inner product of the second and third block rows of (1.6.23) and (1.6.24) must be equal, i.e.,

$$\begin{aligned}&\left[\bar{F}_i^*\bar{U}_i \quad \bar{F}_i^*\tilde{U}_i+\tilde{F}_i^*U_i \quad H^*\right](\bar{D}_i^{-1} \oplus D_i^{-1} \oplus J)\ \left[\tilde{L}_i \quad L_i \ \ 0\right]^* \\ &= \left[\bar{F}_i^*\bar{U}_i \quad \tilde{U}_i \quad H_i^*\right](\bar{D}_i^{-1} \oplus D_i^{-1} \oplus J)\left[\tilde{L}_i \quad F_iL_i \quad G_i\right]^*.\end{aligned}$$

This implies that

$$\bar{F}_i^*\,\tilde{U}_i\,D_i^{-1}\,L_i^* + \tilde{F}_i^*\,U_i\,D_i^{-1}\,L_i^* = \tilde{U}_i\,D_i^{-1}\,L_i^*\,F_i^* + H_i^*\,J\,G_i^*\,.$$

Equating the first columns on both sides leads to the equation

$$\bar{F}_i^*\,\bar{u}_i + \varphi_i^* = \bar{u}_i\,f_i^* + H_i^*\,J\,g_i^*\ ,$$

which validates (1.10.7). The nondegeneracy condition (1.10.1) ensures that (1.10.7) has a unique solution for $\bar{u}_i$.

Finally, by omitting the irrelevant columns and rows of the pre- and postarrays (i.e., those rows and columns that remain unchanged), we can write

$$\left[\begin{array}{c|c} F_i\,l_i & G_i \\ \hline u_i & \begin{matrix} H_i^* \\ 0 \end{matrix} \end{array}\right]\Sigma_i \ = \ \left[\begin{array}{c|c} l_i & \begin{matrix} 0 \\ G_{i+1} \end{matrix} \\ \hline \bar{F}_{i+1}^*\,u_i & H_{i+1}^* \end{array}\right], \tag{1.10.16}$$

where Σ_i is a submatrix of Ω_i as in (1.6.27). This establishes (1.10.8). Also, by using the parameters of Σ_i shown in Lemma 1.6.4, we obtain the generator recursion relating H_i^* and H_{i+1}^* (as stated after Alg. 1.10.2).

1.10.4 An Example

We return to the rotation example we considered in Sec. 1.6.6 with $n=3$ and $r=2$ and show how to incorporate the procedure for inverting R as well. In this case the pre- and postarrays will have the following zero-patterns:

$$\begin{bmatrix} FL & G \\ U & 0 \end{bmatrix} = \left[\begin{array}{ccc|cc} * & 0 & 0 & * & * \\ * & * & 0 & * & * \\ * & * & * & * & * \\ \hline 1 & * & * & 0 & 0 \\ 0 & 1 & * & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right], \quad \begin{bmatrix} L & 0 \\ F^*U & H^* \end{bmatrix} = \left[\begin{array}{ccc|cc} d_0 & 0 & 0 & 0 & 0 \\ * & d_1 & 0 & 0 & 0 \\ * & * & d_2 & 0 & 0 \\ \hline f_0^* & * & * & * & * \\ 0 & f_1^* & * & * & * \\ 0 & 0 & f_2^* & * & * \end{array}\right].$$

Using (1.6.7) for $i = 0$, viz.,

$$(I_n - f_0^* F)l_0 = GJg_0,$$

we can determine the first column l_0 of L and, consequently, the first column of FL. In this way, all the entries of the first column of the prearray are completely known. Also, the last two columns of the prearray are known since they are determined by G. Hence, the $(1, 2)$ block entry of the prearray (i.e., the top row of G) can be eliminated by pivoting with the top entry of the first column of FL. As a result, the first column of the prearray and its last two columns are linearly combined to yield the intermediate array shown below after the transformation Ω_0. The second and third columns of the prearray remain unchanged.

$$\left[\begin{array}{ccc|cc} \boxed{*} & 0 & 0 & \boxed{*} & \boxed{*} \\ * & * & 0 & * & * \\ * & * & * & * & * \\ \hline 1 & * & * & 0 & 0 \\ 0 & 1 & * & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right] \xrightarrow{\Omega_0} \left[\begin{array}{ccc|cc} d_0 & 0 & 0 & 0 & 0 \\ * & \boxed{*} & 0 & \boxed{*} & \boxed{*} \\ * & * & * & * & * \\ \hline f_0^* & * & * & * & * \\ 0 & 1 & * & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right]$$

$$\xrightarrow{\Omega_1} \left[\begin{array}{ccc|cc} d_0 & 0 & 0 & 0 & 0 \\ * & d_1 & 0 & 0 & 0 \\ * & * & \boxed{*} & \boxed{*} & \boxed{*} \\ \hline f_0^* & * & * & * & * \\ 0 & f_1^* & * & * & * \\ 0 & 0 & 1 & 0 & 0 \end{array}\right] \xrightarrow{\Omega_2} \left[\begin{array}{ccc|cc} d_0 & 0 & 0 & 0 & 0 \\ * & d_1 & 0 & 0 & 0 \\ * & * & d_2 & 0 & 0 \\ \hline f_0^* & * & * & * & * \\ 0 & f_1^* & * & * & * \\ 0 & 0 & f_2^* & * & * \end{array}\right].$$

We now proceed in a similar fashion to the next step. Using (1.6.7) for $i = 1$, viz.,

$$(I_{n-1} - f_1^* F_1)l_1 = G_1 J g_1,$$

we determine the l_1 and, consequently, the second column of FL. Likewise, using (1.10.7) for $i = 1$ (in this case $\bar{F}_1 = f_0$),

$$(f_0^* - f_1^*)\bar{u}_1 = H_1^* J g_1^* - \varphi_1^*,$$

we determine $\bar{u}_1$. The second transformation Ω_1 can now be performed as shown above to yield G_2 and H_2, and so on.

The rectangular boxes mark the entries to be eliminated at each step of the recursion by using elementary $(D^{-1} \oplus J)$-unitary transformations. The square boxes mark the position of the pivot elements. The ultimate result of the recursion is that the $(1, 2)$ block of the prearray is eliminated row by row ("reduction procedure"), while the $(2, 2)$ block is filled up with nonzero elements ("construction procedure").

1.10.5 Proper Form of the Algorithm

Analogous to what we did in Sec. 1.7 for the recursion for the generators G_i, we can reduce the recursion for the H_i^* in Alg. 1.10.2 into proper form.

Assume again that $g_iJg_i^* > 0$ and let Θ_i be a J-unitary matrix that rotates g_i to the form (1.7.1). Then it can be verified easily that the expression (1.10.7) for $\bar{u}_i$ reduces to

$$(\bar{F}_i^* - f_i^* I_i)\,\bar{u}_i = H_i^*\Theta_i \begin{bmatrix} \delta_i^* \\ 0 \end{bmatrix} - \varphi_i^*,$$

while the recursion for H_i^* reduces to

$$H_{i+1}^* = \begin{bmatrix} H_i^* \\ 0 \end{bmatrix}\Theta_i\begin{bmatrix} 0 & 0 \\ 0 & I_{r-1} \end{bmatrix} + \begin{bmatrix} \Psi_i & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} H_i^* \\ 0 \end{bmatrix}\Theta_i\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \phi_i \\ 1 \end{bmatrix}\begin{bmatrix} \sigma_i & 0 \end{bmatrix},$$

where we defined $\sigma_i \triangleq \delta_i/d_i$. The above equation has the following interpretation:

1. Multiply H_i^* by Θ_i.
2. Multiply the *first* column of $H_i^*\Theta_i$ by Ψ_i and keep the rest of the columns unaltered.
3. Attach a zero row to the bottom of the array.
4. Add the correction term $\sigma_i[\ \phi_i^* \ \ 1\ ]^*$ to the *first* column.

Note that initially H_i^* is in proper form. Multiplying the array by Θ_i will destroy this properness (see Fig. 1.5). After attaching a zero row to the bottom of the matrix and adding a correction term to the first column, the resulting matrix H_{i+1}^* will emerge in proper form again.

$$H_i^* = \begin{bmatrix} * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \\ * & 0 & 0 \end{bmatrix} \xrightarrow{\text{Steps } 1-2} \begin{bmatrix} * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \\ * & * & * \end{bmatrix}$$

$$\xrightarrow{\text{Step } 3} \begin{bmatrix} * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \\ * & * & * \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{Step } 4} \begin{bmatrix} * & * & * \\ \vdots & \vdots & \vdots \\ * & * & * \\ * & * & * \\ * & 0 & 0 \end{bmatrix} = H_{i+1}^*$$

Figure 1.5. Proper form of the generator recursion for inversion.

When, on the other hand, $g_iJg_i^* < 0$ we let Θ_i be a J-unitary matrix that rotates g_i to the form (1.7.4). Then the expression for (1.10.7) $\bar{u}_i$ becomes

$$(\bar{F}_i^* - f_i^* I_i)\,\bar{u}_i = -H_i^*\Theta_i \begin{bmatrix} 0 \\ \delta_i^* \end{bmatrix} - \varphi_i^*,$$

and the recursion for H_i^* now reduces to

$$H_{i+1}^* = \begin{bmatrix} H_i^* \\ 0 \end{bmatrix}\Theta_i\begin{bmatrix} I_{r-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \Psi_i & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} H_i^* \\ 0 \end{bmatrix}\Theta_i\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} \phi_i \\ 1 \end{bmatrix}\begin{bmatrix} 0 & \sigma_i \end{bmatrix}.$$

This equation has the following interpretation:

1. Multiply H_i^* by Θ_i.
2. Multiply the *last* column of $H_i^*\Theta_i$ by Ψ_i and keep the rest of the columns unaltered.
3. Attach a zero row to the bottom of the array.
4. Add the correction term $\sigma_i[\ \phi_i^*\ \ 1\ ]^*$ to the *last* column.

1.10.6 Application to Pick Matrices

We reconsider the case of Pick matrices (1.8.4), which, as discussed at the end of Sec. 1.8.2, did not yield to the embedding technique for the factorization of the inverse matrix. More specifically, by starting with the matrix P in (1.8.4) and by using $F = \text{diag}\{f_0, f_1, \ldots, f_{n-1}\}$, we saw that the extended matrix

$$M = \begin{bmatrix} -P & I \\ I & 0 \end{bmatrix}$$

did not have low displacement rank with respect to $M - (F \oplus F)M(F \oplus F)^*$ since $I - FF^*$ is full rank in general.

However, the fast inversion algorithm just derived overcomes this difficulty. Indeed, according to (1.8.5) we have

$$G = \begin{bmatrix} u_0 & v_0 \\ u_1 & v_1 \\ \vdots & \vdots \\ u_{n-1} & v_{n-1} \end{bmatrix}, \quad F = \begin{bmatrix} f_0 & & & \\ & f_1 & & \\ & & \ddots & \\ & & & f_{n-1} \end{bmatrix}.$$

Now Lemma 1.5.1 implies that

$$R^{-1} - F^* R^{-1} F = H^* J H$$

for some matrix $H \in \mathbb{C}^{r\times n}$. This means that R^{-1} is also a Pick matrix,

$$R^{-1} = \left[\ \frac{a_i a_j^* - b_i b_j^*}{1 - f_i^* f_j}\ \right]_{i,j=0}^{n-1},$$

where $[\ a_i\ \ b_i\ ]$ denotes the ith row of H^*. It further follows from the diagonal structure of F that

$$\begin{aligned} \Phi_i &= \text{diag}\left\{\ 0\ \ \frac{f_{i+1}-f_i}{1-f_i^* f_{i+1}}\ \ \frac{f_{i+2}-f_i}{1-f_i^* f_{i+2}}\ \ \cdots\ \ \frac{f_{n-1}-f_i}{1-f_i^* f_{n-1}}\ \right\}, \\ \Psi_i &= \text{diag}\left\{\ \frac{1-f_i f_0^*}{f_0^*-f_i^*}\ \ \frac{1-f_i f_1^*}{f_1^*-f_i^*}\ \ \cdots\ \ \frac{1-f_i f_{i-1}^*}{f_{i-1}^*-f_i^*}\ \right\}, \\ \phi_i &= 0. \end{aligned}$$

The generator matrix H can then be determined by resorting to the fast inversion Alg. 1.10.2 (or to the proper form of Sec. 1.10.5).

1.10.7 The Degenerate Case

The derivation of the fast inversion algorithm of Sec. 1.10.2 was based on the nondegeneracy condition (1.10.1), viz., that the diagonal entries of F are distinct. This condition ensured that (1.10.7) had a unique solution for $\bar{u}_i$.

In this section we show how to relax the nondegeneracy assumption (1.10.1). We do so by focusing on the case when F consists of a single Jordan block so that $f_0 = f_1 = \cdots = f_{n-1}$ holds:

$$F = \begin{bmatrix} f_0 & & & & \\ 1 & f_0 & & & \\ & 1 & f_0 & & \\ & & \ddots & \ddots & \\ & & & 1 & f_0 \end{bmatrix} = Z + f_0 I \tag{1.10.17}$$

with $1-|f_0|^2 \neq 0$. This case clearly includes the special choice $F = Z$ (which corresponds to $f_0 = 0$). The argument we give here, however, can be extended easily to handle the more general case of a matrix F with multiple (or even repeated) Jordan blocks.

For a matrix F as in (1.10.17), it is easy to verify from (1.10.7) that all the entries of $\bar{u}_i$ can be determined uniquely from (1.10.7), except for the top entry of $\bar{u}_i$. We shall denote this top entry by ξ_i. This means that the fast inversion algorithm that we derived in Sec. 1.10 almost completely identifies the upper triangular factor U with the exception of its top row:

$$U = \begin{bmatrix} 1 & \xi_1 & \xi_2 & \cdots & \xi_{n-1} \\ & 1 & \times & \cdots & \times \\ & & 1 & \cdots & \times \\ & & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}.$$

In the above expression, the symbol $\times$ denotes known entries. The unknown parameters $\{\xi_i\}$ can be identified by resorting to the fundamental equality (1.6.13), which provides an upper triangular system of linear equations in the $\{\xi_i\}$:

$$\begin{bmatrix} 1 & \xi_1 & \xi_2 & \cdots & \xi_{n-1} \end{bmatrix} D^{-1}L^* = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{1.10.18}$$

Since the matrices $\{D, L\}$ can be determined without ambiguity from the recursions of the generalized Schur algorithm for the direct factorization problem, we can therefore use the above linear system of equations and determine the $\{\xi_i\}$.

More specifically, let L_{i+1} and D_{i+1} denote the leading $(i+1) \times (i+1)$ submatrices of L and D, respectively. Given $\{L_{i+1}, D_{i+1}\}$ and $\{\xi_1, \ldots, \xi_{i-1}\}$, we can determine ξ_i by solving

$$\begin{bmatrix} 1 & \xi_1 & \xi_2 & \cdots & \xi_i \end{bmatrix} D_{i+1}^{-1}L_{i+1}^* = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{1.10.19}$$

Therefore, the only additional computation relative to Alg. 1.10.2 is the need to determine the top entries $\{\xi_i\}$ of the successive $\bar{u}_i$ as explained above. More can be said about the inversion algorithm in the degenerate case. We omit the discussion here and refer instead to [BSLK96], [BKLS98a], [BKLS98b].

1.11 NON-HERMITIAN TOEPLITZ-LIKE MATRICES

The derivation in the earlier sections was primarily devoted to Hermitian Toeplitz-like matrices R that satisfy displacement equations of the form $R - FRF^* = GJG^*$.

As mentioned before, we can also treat non-Hermitian Toeplitz-like matrices. Such matrices admit a triangular factorization of the form $R = LD^{-1}U$, where L is lower triangular and U is upper triangular with identical diagonal entries, and which are equal to those of D. In the Hermitian case, $U = L^*$. In what follows, we denote the (nonzero parts of the) columns and *rows* of L and U by $\{l_i, u_i\}$, respectively. (Observe that we are now using the letter U to denote the upper triangular factor of R and not of R^{-1}. We are also writing u_i to denote a row of U.)

We shall not repeat the derivation of the Schur algorithm in this context but will only state one of its forms; derivations can be found in [KS95a]. It will be noted that the general form of the recursions is still very similar to what we had in the Hermitian case, except that now we need to propagate two generator matrices. For reasons of space, we present only the more compactly described (explicit) equation forms.

Algorithm 1.11.1 (Non-Hermitian Toeplitz-Like Matrices). *Consider an $n \times n$ strongly regular matrix R with displacement structure*

$$R - FRA^* = GB^* ,$$

where F and A are $n \times n$ lower triangular matrices whose diagonal entries are denoted by $\{f_i, a_i\}$, respectively, and G and B are $n \times r$ generator matrices. It is further assumed that

$$1 - f_j a_i^* \neq 0 \quad \textit{for all } i, j.$$

Then the successive Schur complements of R satisfy

$$R_i - F_i R_i A_i^* = G_i B_i^*,$$

where $\{F_i, A_i\}$ are the submatrices obtained after deleting the first row and column of the corresponding $\{F_{i-1}, A_{i-1}\}$ and G_i and B_i are $(n-i) \times r$ generator matrices that satisfy the following recursions: start with $G_0 = G, B_0 = B$, $F_0 = F, A_0 = A$ and repeat for $i \geq 0$:

$$\begin{aligned} \begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} &= \left[G_i + (\Phi_i - I_{n-i}) G_i \frac{b_i^* g_i}{g_i b_i^*} \right] \Theta_i, \\ \begin{bmatrix} 0 \\ B_{i+1} \end{bmatrix} &= \left[B_i + (\Psi_i - I_{n-i}) B_i \frac{g_i^* b_i}{b_i g_i^*} \right] \Gamma_i, \end{aligned} \tag{1.11.1}$$

where Θ_i and Γ_i are arbitrary matrices that satisfy $\Theta_i \Gamma_i^* = I$, g_i and b_i are the top rows of G_i and B_i, respectively, and

$$\Phi_i = (F_i - f_i I)(I - a_i^* F_i)^{-1}, \qquad \Psi_i = (A_i - a_i I)(I - f_i^* A_i)^{-1}. \tag{1.11.2}$$

The triangular factors are given by

$$l_i = (I_{n-i} - a_i^* F_i)^{-1} G_i b_i^*, \quad u_i = g_i B_i^* (I_{n-i} - f_i A_i^*)^{-1}, \quad d_i = g_i b_i^* / (1 - f_i a_i^*).$$

◇

Array forms of these recursions are also treated in [KS95a] and they can be described briefly as follows. Choose the parameters Θ_i and Γ_i to reduce g_i and b_i to the forms

$$g_i \Theta_i = \begin{bmatrix} 0 & \bar{x}_i & 0 \end{bmatrix} \quad \text{and} \quad b_i \Gamma_i = \begin{bmatrix} 0 & \bar{y}_i & 0 \end{bmatrix},$$

respectively, where the nonzero entries $\bar{x}_i$ and $\bar{y}_i$ are in the same column position, say, the jth position. (Generalizations of the Givens and Householder transformations can be obtained for finding $\{\Theta_i, \Psi_i\}$—see Sec. 4.4.3 of [KS95a].) Then it can be verified that the generator recursions collapse to

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = \Phi_i G_i \Theta_i \begin{bmatrix} 0_{j\times j} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + G_i\Theta_i \begin{bmatrix} I_j & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_{r-j-1} \end{bmatrix},$$
$$\begin{bmatrix} 0 \\ B_{i+1} \end{bmatrix} = \Psi_i B_i \Gamma_i \begin{bmatrix} 0_{j\times j} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + B_i\Gamma_i \begin{bmatrix} I_j & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & I_{r-j-1} \end{bmatrix}, \qquad (1.11.3)$$

where $d_i = (\bar{x}_i J_{jj} \bar{y}_i^*)/(1 - f_i a_i^*)$, and

$$l_i = (I_{n-i} - a_i^* F_i)^{-1} G_i \Theta_i J \begin{bmatrix} 0 \\ \bar{y}_i^* \\ 0 \end{bmatrix}, \quad u_i = \begin{bmatrix} 0 & \bar{x}_i & 0 \end{bmatrix} J \Gamma_i^* B_i^* (I - f_i A_i^*)^{-1}.$$

These algorithms are useful in studying (unconstrained) rational interpolation problems (see Sec. 1.14.3 and [BSK94], [BSK99]).

1.12 SCHUR ALGORITHM FOR HANKEL-LIKE MATRICES

To round out our discussions of generalized Schur algorithms, we finally consider the case of strongly regular Hankel-like matrices. As with the Toeplitz-like structure, the Hankel-like structure is also preserved under Schur complementation. Similar arguments will show that the following recursions hold—they are special cases of the general algorithm first derived in [KS91], [Say92] (see also [SK95a], [KS95b], Sec. 7.2.5 of [KS95a]), and are used in [GKO95].

Algorithm 1.12.1 (Schur Algorithm for Hankel-Like Matrices). *Consider an $n \times n$ strongly regular Hankel-like matrix that satisfies*

$$FR - RA^* = GB^*, \qquad (1.12.1)$$

where the diagonal entries of the lower triangular matrices F and A are denoted by $\{f_i, a_i\}$, respectively, and satisfy

$$f_i - a_j^* \neq 0 \quad \textit{for all } i, j. \qquad (1.12.2)$$

Then the successive Schur complements R_i satisfy

$$F_i R_i - R_i A_i^* = G_i B_i^*, \qquad (1.12.3)$$

where F_i and A_i are the submatrices obtained after deleting the first row and column of F_{i-1} and A_{i-1}, respectively, and G_i and B_i are $(n-i) \times r$ generator matrices that satisfy, along with l_i and u_i (the first column and row of R_i), the following recursions:

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = \left\{G_i - l_i d_i^{-1} g_i\right\} \Theta_i, \qquad (1.12.4)$$

$$\begin{bmatrix} 0 \\ B_{i+1} \end{bmatrix} = \left\{B_i - u_i^* d_i^{-*} b_i\right\} \Gamma_i, \qquad (1.12.5)$$

where Θ_i and Γ_i are arbitrary parameters that satisfy $\Theta_i\Gamma_i^* = I$. Moreover,

$$d_i = \frac{g_i b_i^*}{f_i - a_i^*}, \tag{1.12.6}$$

$$l_i = (F_i - a_i^* I_{n-i})^{-1} G_i b_i^*, \tag{1.12.7}$$

$$u_i = g_i B_i^* (f_i I_{n-i} - A_i^*)^{-1}. \tag{1.12.8}$$

◇

Remark 1. Array forms for these equations also exist and are discussed in [KS95a].

Remark 2. The condition (1.12.2) is necessary to guarantee a unique solution R of the displacement equation (1.12.1). It further guarantees that the expressions (1.12.6)–(1.12.8) are well defined and uniquely determine the quantities $\{d_i, l_i, u_i\}$. When (1.12.2) is violated, so that the inverses $(F_i - a_i^* I_{n-i})^{-1}$ and $(f_i I_{n-i} - A_i^*)^{-1}$ in (1.12.7) and (1.12.8) need not exist, then we need to determine the $\{d_i, l_i, u_i\}$ by solving the equations

$$(f_i - a_i^*)d_i = g_i b_i^*\ , \tag{1.12.9}$$

$$(F_i - a_i^* I_{n-i})l_i = G_i b_i^*\ , \tag{1.12.10}$$

$$u_i(f_i I_{n-i} - A_i^*) = g_i B_i^*\ . \tag{1.12.11}$$

The nonsingularity of $(F_i - a_i^* I_{n-i})$ and $(f_i I_{n-i} - A_i^*)$ would imply that these equations have many solutions $\{d_i, l_i, u_i\}$. For this reason, additional information (often known as coupling numbers) is needed to fully recover the $\{d_i, l_i, u_i\}$. These issues are not of major concern to us here since the fundamental equations (1.12.4) and (1.12.5) will be the same. More detailed discussions can be found in [KO96], [KO98] (see also [KS95a]).

1.13 STRUCTURED MATRICES AND PIVOTING

An issue that arises in the study of fast factorization algorithms is their numerical stability in finite precision implementations. It was mentioned in Sec. 1.6.2 that the Schur reduction procedure, which underlies the generalized Schur algorithm, is equivalent to the Gaussian elimination procedure, because the latter can be rewritten as

$$R = \begin{bmatrix} 1 & 0 \\ t_0 d_0^{-1} & I_{n-1} \end{bmatrix} \begin{bmatrix} d_0 & t_0^* \\ 0 & R_1 \end{bmatrix}.$$

Thus the generalized Schur algorithm amounts to combining Gaussian elimination with structure. Now it is well known (see, e.g., [GV96], [TB97], and Sec. 4.2.1) that Gaussian elimination in its purest form is numerically unstable (meaning that the error in the factorization $\hat{L}\hat{D}^{-1}\hat{L}^*$ can be quite large, where $\hat{L}$ and $\hat{D}$ denote the computed L and D, respectively). The instability often can be controlled by resorting to pivoting techniques, viz., by permuting the order of the rows, and perhaps columns, of the matrices before the Gaussian elimination steps.

In what is known as *complete pivoting*, a permutation matrix P_k is chosen at each iteration k so as to bring the maximal magnitude entry in the entire matrix R_k to the pivotal $(0,0)$th position. Such a procedure is computationally intensive since it requires many comparisons. A less-demanding procedure is *partial pivoting*. In this case, the permutation matrix P_k is chosen so as to bring at the kth step the maximal magnitude entry of the first column of R_k to the pivotal $(0,0)$th position. (Although partial pivoting often performs satisfactorily, there are several examples where the numerical accuracy of the factorization can still be poor—see, e.g., [Hig96].) In either case, complete or partial pivoting leads to the triangular factorization of a permuted verion of R, say, $PR = LD^{-1}L^*$.

1.13.1 Incorporating Pivoting into Generalized Schur Algorithms

Unfortunately, pivoting can destroy matrix structure and thus can lead to a loss in computational efficiency. There are, however, matrices whose structure is unaffected by partial pivoting, e.g., Vandermonde and Cauchy matrices or even Cauchy-like matrices, as first noted and exploited by Heinig [Hei95].

Recall that Cauchy-like matrices are special cases of the class of Hankel-like matrices in that they satisfy displacement equations of the form

$$FR - RA^* = GB^*, \tag{1.13.1}$$

where $\{F, A\}$ are now *diagonal.* Let P denote a permutation matrix that permutes the rows of R. Then $PP^T = I$ and we note that

$$(PFP^T) \cdot (PR) - (PR) \cdot A^* = (PG) \cdot B^*,$$

where PFP^T is still diagonal. We therefore see that the permuted matrix PR is still Cauchy-like with respect to the displacement operators $\{PFP^T, A\}$ and has generator matrices $\{PG, B\}$. In other words, partial pivoting does not destroy the Cauchy-like structure.

More generally, partial pivoting does not destroy the matrix structure as long as some displacement operators are diagonal, e.g.,

$$\begin{aligned} R - FRA^* &= GB^* \quad \text{with } F \text{ diagonal only}, && (1.13.2)\\ FR - RA^* &= GB^* \quad \text{with } F \text{ diagonal only}, && (1.13.3)\\ \Omega R \Delta^* - FRA^* &= GB^* \quad \text{with } \Omega \text{ and } F \text{ diagonal only}. && (1.13.4)\end{aligned}$$

However, for definiteness, we continue our discussion here by focusing on the Cauchy-like case; similar arguments apply in the other cases—see, e.g., Sec. 2.10.

The following algorithm now follows immediately from Alg. 1.12.1 (and is used in [GKO95]); it incorporates partial pivoting into the generalized Schur algorithm for Cauchy-like matrices.

Algorithm 1.13.1. (Schur Algorithm for Cauchy-Like Matrices with Pivoting). *Consider an $n \times n$ strongly regular Cauchy-like matrix that satisfies*

$$FR - RA^* = GB^* \tag{1.13.5}$$

with diagonal $\{F, A\}$ whose entries satisfy $f_i - a_j^* \neq 0$ for all i, j. Start with $G_0 = G$, $B_0 = B$, $F_0 = F$, $A_0 = A$ and repeat for $i = 0, 1, \ldots$:

1. *Determine $\{l_i, u_i\}$ from*

$$\begin{aligned} l_i &= (F_i - a_i^* I_{n-i})^{-1} G_i b_i^*\,, && (1.13.6)\\ u_i &= g_i B_i^* (f_i I_{n-i} - A_i^*)^{-1}\,. && (1.13.7)\end{aligned}$$

2. *Determine the position of the maximal magnitude entry of l_i, say, at the jth position, and let P_i be the permutation matrix that exchanges it with the top entry of l_i. Let d_i be equal to this maximal entry.*

3. *Likewise, exchange the $(0,0)$th diagonal entry of F_i with its (j,j)th diagonal entry. Exchange also the first and the jth rows of G_i. At the end of this step, all three quantities $\{l_i, F_i, G_i\}$ have undergone permutation, but we continue to denote them by the same symbols.*

4. *Now update the generator matrices $\{G_i, B_i\}$ using*

$$\begin{bmatrix} 0 \\ G_{i+1} \end{bmatrix} = \left\{G_i - l_i d_i^{-1} g_i\right\} \Theta_i, \tag{1.13.8}$$

$$\begin{bmatrix} 0 \\ B_{i+1} \end{bmatrix} = \left\{B_i - u_i^* d_i^{-*} b_i\right\} \Gamma_i, \tag{1.13.9}$$

where Θ_i and Γ_i are arbitrary parameters that satisfy $\Theta_i \Gamma_i^* = I$ (e.g., $\Theta_i = \Gamma_i = I$).

◇

At the end of the procedure we obtain the triangular factorization

$$PR = LD^{-1}L^*,$$

where P is the combination of all the individual permutation matrices

$$P = P_0 \begin{bmatrix} 1 & \\ & P_1 \end{bmatrix} \begin{bmatrix} I_2 & \\ & P_2 \end{bmatrix} \cdots .$$

Remark 1. For Hermitian Cauchy-like matrices R, viz., those that satisfy $FR + RF^* = GJG^*$, partial pivoting destroys the symmetry. In such cases, one usually employs *diagonal pivoting*—see [KO98].

Remark 2. For the alternative cases (1.13.2)–(1.13.4), we simply incorporate steps similar to steps 2 and 3 above into the corresponding recursions (in array form or not).

1.13.2 Transformations to Cauchy-Like Structures

As noted above, incorporation of partial pivoting into the generalized Schur algorithm is possible for Cauchy-like, Hankel-like, Toeplitz-like, and even generalized structures with certain diagonal operators F or $\{F, \Omega\}$. But what about matrices not in these classes? Heinig had the nice idea that one could first transform them into matrices to which pivoting could be applied; in particular, he proposed transforming them to Cauchy-like matrices [Hei95]. (Transformations between different kinds of structured matrices were perhaps first proposed, in a different context, by Bini and Pan (see, e.g., [BP94] and the references therein).) We illustrate the procedure in the Toeplitz case.

Thus consider an $n \times n$ non-Hermitian Toeplitz matrix T. As mentioned at the begining of Sec. 1.3, there are many forms of displacement structure even for the same matrix. In particular, T also has displacement rank 2 with respect to the displacement operation

$$\nabla_{\{Z_1, Z_{-1}\}} R = Z_1 R - R Z_{-1} = GB^*, \tag{1.13.10}$$

where Z_ϕ denotes the ϕ-circulant matrix

$$Z_\phi = \begin{bmatrix} 0 & 0 & \cdots & 0 & \phi \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}.$$

Heinig [Hei95] showed that the above displacement equation can be transformed to Cauchy-like form as follows. The matrix Z_ϕ can be diagonalized by the scaled discrete Fourier matrix

$$\mathcal{F} = \frac{1}{\sqrt{n}}\left[\omega^{ij}\right]_{i,j=0}^{n-1},$$

with ω denoting the primitive nth root of unity. More specifically, it holds that

$$Z_\phi = (D_\phi^{-1}\mathcal{F}^*)D_{Z_\phi}(\mathcal{F}\cdot D_\phi) \tag{1.13.11}$$

with

$$D_{Z_\phi} = \text{diag}\,\{\xi\omega^i\}_{i=0}^{n-1}, \qquad D_\phi = \text{diag}\,\{\xi^i\}_{i=0}^{n-1},$$

where ξ is an arbitrary complex number satisfying $\xi^n = \phi$. Now define the transformed matrix

$$R = \mathcal{F}TD_{-1}^*\mathcal{F}^*.$$

Then R satisfies the Cauchy-like displacement equation

$$D_1R - RD_{-1} = (\mathcal{F}G)(BD_{-1}^*\mathcal{F}^*).$$

Note that R is in general complex valued even when T is real valued. This increases the constant factors in the operation count due to the need for complex arithmetic. A study of this procedure, with several examples and extensions, can be found in [GKO95].

1.13.3 Numerical Issues

While the transformation-and-pivoting technique of the last two sections can be satisfactory in many situations, it still suffers from two problems. First, the method applies only to a fixed matrix size $n \times n$, and the whole solution has to be repeated if the size of the matrix is increased even by 1. Second, the procedure can still pose numerical problems because partial pivoting by itself is not sufficient to guarantee numerical stability even for slow algorithms (see, e.g., the discussion and examples in Ch. 4 by Brent).

A more direct approach to the numerical stability of the generalized Schur algorithm is to examine the steps of the algorithm directly and to stabilize them without resorting to transformations among matrix structures. This is pursued in Chs. 2, 3, and 4. For all practical purposes, the main conclusion of Chs. 2 and 4 is that the generalized Schur algorithm is numerically stable for a large class of positive-definite structured matrices. In Ch. 3, it is further shown how this conclusion can be extended to indefinite structured matrices.

Chapter 4 by Brent provides, among other results, an overview of the conclusions in [BBHS95]. This reference studied the stability of the generalized Schur algorithm for the subclass of positive-definite quasi-Toeplitz structured matrices ($F = Z$ and displacement rank 2) and established that the triangular factorization provided by the algorithm is in effect asymptotically stable regardless of the hyperbolic rotations. In [SD97b] it was further shown that for higher displacement ranks, special care is needed while implementing the rotations in order to still guarantee stable factorizations.

The results in [BBHS95] motivated Chandrasekaran and Sayed [CS96] to study the stability of the generalized Schur algorithm for a wider class of matrices, viz., positive-definite matrices R for which the shift structure matrix Z is replaced by a lower triangular F (as in the definition (1.3.21)). Their conclusions are reviewed in Ch. 2, where it is shown that, for all practical purposes, by incorporating a few enhancements to the algorithm, it yields backward-stable factorizations for a wide class of structured matrices.

This is a reassuring conclusion. However, it applies only to positive-definite structured matrices. In [CS98], Chandrasekaran and Sayed further showed how to employ the embedding ideas proposed in [KC94] to develop fast backward-stable solvers for linear systems of equations, say, $Tx = b$, with possibly indefinite and even nonsymmetric structured coefficient matrices T (see Ch. 3). This is achieved by transforming a problem that involves a nonsymmetric or indefinite structured matrix into an equivalent problem that involves sign-definite matrices only (either positive definite or negative definite). This is possible by introducing the larger matrix

$$M = \begin{bmatrix} T^*T & T^* \\ T & 0 \end{bmatrix}$$

and by observing that, regardless of T, the matrix M is always Hermitian. Moreover, its leading block is positive definite and the Schur complement with respect to it is negative definite (in fact, equal to $-I$). When T is structured, the matrix M also has structure and its factorization can be carried out efficiently by means of the generalized Schur algorithm. By factoring M fast and stably, the solution x of $Tx = b$ can be determined fast and stably. These results are reviewed in Ch. 3.

1.14 SOME FURTHER ISSUES

Although a wide range of results has already been addressed in this chapter, there are still several unmentioned results and applications. We give a brief outline of a few of these items here, some of which are treated at greater length in later chapters. Other items are covered in the article [KS95a].

1.14.1 Incorporating State-Space Structure

A very powerful and well-studied structure in system theory is state-space structure. A typical scenario is the following. We have a stochastic process $\{\mathbf{y}_i, i \geq 0\}$ having a model of the form

$$\begin{aligned} \mathbf{x}_{i+1} &= F_i\mathbf{x}_i + G_i\mathbf{u}_i, \\ \mathbf{y}_i &= H_i\mathbf{x}_i + \mathbf{v}_i, \end{aligned}$$

where $\{\mathbf{u}_i, \mathbf{v}_i\}$ are zero-mean white noise processes with covariance matrices

$$E\begin{bmatrix} \mathbf{u}_i \\ \mathbf{v}_i \end{bmatrix}\begin{bmatrix} \mathbf{u}_i^* & \mathbf{v}_i^* \end{bmatrix} = \begin{bmatrix} Q_i & S_i \\ S_i^* & R_i \end{bmatrix}\delta_{ij}.$$

The initial state, $\mathbf{x}_0$, is also assumed to be a zero-mean random variable with variance Π_0 and uncorrelated with $\{\mathbf{u}_i, \mathbf{v}_i\}$, i.e.,

$$E\mathbf{x}_0\mathbf{x}_0^* = \Pi_0, \quad E\mathbf{u}_i\mathbf{x}_0^* = 0, \quad E\mathbf{v}_i\mathbf{x}_0^* = 0, \quad i \geq 0.$$

The processes are vector valued, with $\{\mathbf{u}_i\}$ being q-dimensional, the states $\mathbf{x}_i$ being n-dimensional, and the measurement noise $\{\mathbf{v}_i\}$ and the output $\{\mathbf{y}_i\}$ being p-dimensional. It is assumed that $\{q, n, p\}$ are known, as are all the matrices $\{F_i, G_i, H_i, \Pi_0, Q_i, R_i, S_i\}$.

The solutions of many different problems associated with such models are closely related (see, e.g., [KSH99]) to the triangular factorization of the covariance matrix of the output process $\{\mathbf{y}_i\}$, say,

$$R_y = E\mathbf{y}\mathbf{y}^*, \quad \mathbf{y} \triangleq \text{col}\{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{N-1}\}.$$

Although R_y is $N \times N$, and N is often very large, the fact that it is the covariance matrix of a process with an n-dimensional state-space model (where generally $n \ll N$) means that the triangular factorization should take fewer than the $O(N^3)$ flops required for an arbitrary $N \times N$ matrix. In fact, the flop count is $O(Nn^3)$, which is achieved via a so-called discrete-time Riccati recursion for an associated $n \times n$ matrix P_i; this is shown in books on state-space estimation and control (see, e.g., [KSH99] and the references therein).

When the model parameters are time invariant, it turns out that the $N \times N$ matrix R_y has displacement structure, with displacement rank $r \leq n$. In this case, the flop count can be reduced to $O(Nn^2)$ by developing an appropriate generalized Schur algorithm (see, e.g., [SLK94a]). The time-invariance assumption can actually be relaxed somewhat to allow a structured form of time variation, which is encountered, for example, in problems of adaptive filtering (see, e.g., [SK94a], [SK94b]). When displacement structure is present, the Riccati recursion is replaced by certain so-called generalized Chandrasekhar recursions, first introduced in [Kai73] and [MSK74]. The survey article [KS95a] gives an outline of how state-space structure can be combined with displacement structure.

In fact, the above studies inspired a generalization of the definition of displacement structure, for example, using equations of the form

$$\nabla_{F(t)} R(t) \overset{\Delta}{=} R(t) - F(t)R(t-\Delta)F^*(t). \tag{1.14.1}$$

We refer to [SCK94], [SLK94b] for properties and applications of this extension.

The power of state-space representations makes it useful to seek to obtain them from input-output descriptions. For time-variant systems, this has been studied in [DV98]. In Ch. 10 of this volume, Dewilde describes how these ideas can be combined with displacement structure to obtain low-complexity approximations of matrices.

1.14.2 Iterative Methods

Existing methods for the solution of linear systems of equations of the form $Ax = b$ can be classified into two main categories: *direct* methods and *iterative* methods. A direct method or algorithm is primarily concerned with first obtaining the triangular or QR factors of A and then reducing the original equations $Ax = b$ to an equivalent triangular system of equations. The generalized Schur algorithm of this chapter leads to a direct method of solution.

An iterative method, on the other hand, starts with an initial guess for the solution x, say, x_0, and generates a sequence of approximate solutions, $\{x_k\}_{k\geq 1}$. The matrix A itself is involved in the iterations via matrix-vector products, and the major concern here is the speed of convergence of the iterations. To clarify this point, we note that we can rewrite the equation $Ax = b$ in the equivalent form

$$Cx = (C - A)x + b$$

for an arbitrary invertible matrix C. This suggests the following iterative scheme (see, e.g., [GV96]),

$$Cx_{i+1} = (C - A)x_i + b, \quad x_0 = \text{initial guess}. \tag{1.14.2}$$

The convergence of (1.14.2) is clearly dependent on the spectrum of the matrix $I - C^{-1}A$. The usefulness of (1.14.2) from a practical point of view is very dependent on the choice for C. For Toeplitz matrices, Strang [Str86] proposed certain circulant preconditioners C, which allow the use of the FFT technique to carry out the computations in a numerically efficient and parallelizable manner.

A survey of this method, with emphasis on Toeplitz linear equations and many later developments, is provided by Chan and Ng in Ch. 5; closer study of the nullspaces of appropriate displacement operators leads to new families of preconditioners [KO96]. The study of spectral properties of Toeplitz matrices is important in this theory. In Ch. 6, Tilli provides a review of recent results in this area, especially for block Toeplitz matrices. Fast algorithms for block Toeplitz matrices are developed in Ch. 8 by Bini and Meini.

An iterative method that offers faster convergence rates than the above methods is based on the use of Newton iterations. In Ch. 7, Pan et al. describe how displacement structure ideas can be used to speed up the Newton iterations.

1.14.3 Interpolation Problems

Interpolation problems of various types have a long history in mathematics and in circuit theory, control theory, and system theory. Not surprisingly, this rich subject can be approached in many ways and in different settings, often involving a lot of quite abstract operator theory (see, e.g., the monographs [Hel87], [Dym89a], [FF90], [GS94]). For the rational case, we have the somewhat more concrete state-space approach of [BGR90].

In [Say92], [SK92], [SKLC94], a recursive approach to rational analytic interpolation problems has been proposed that relies on the displacement structure framework; it leads to a computationally efficient procedure that avoids matrix inversions. Reference [SKLC94] elaborates on connections with earlier works on the subject.

The basis for the approach of [SKLC94] is the generalized Schur algorithm of this chapter, which leads, as explained in Sec. 1.7.4, to a cascade of J-lossless first-order sections, each of which has an evident interpolation property. This is due to the fact that linear systems have "transmission zeros": certain inputs at certain frequencies yield zero outputs. More specifically, each section of the cascade can be seen to be characterized by a $(p+q) \times (p+q)$ rational transfer matrix, $\Theta_i(z)$, say, that has a left zero-direction vector g_i at a frequency f_i, viz.,

$$g_i\Theta_i(f_i) \equiv \begin{bmatrix} a_i & b_i \end{bmatrix} \begin{bmatrix} \Theta_{i,11} & \Theta_{i,12} \\ \Theta_{i,21} & \Theta_{i,22} \end{bmatrix} (f_i) = 0,$$

which makes evident (with the proper partitioning of the row vector g_i and the matrix function $\Theta_i(z)$) the following interpolation property: $a_i\Theta_{i,12}\Theta_{i,22}^{-1}(f_i) = -b_i$. This suggested to us that one way of solving an interpolation problem is to show how to construct an appropriate cascade so that the local interpolation properties of the elementary sections combine in such a way that the cascade yields a solution to the global interpolation problem. All possible interpolants can then be parametrized by attaching various loads to the right-hand side of the cascade system. Details are provided in [SKLC94], [KS95a], where different kinds of analytic interpolation problems are considered, including the problems of Carathéodory, Nevanlinna–Pick, and Hermite–Fejér. An application to the so-called four-block problem in H^∞-control can be found in [CSK94].

Actually, the arguments can also be extended to the very old class of unconstrained rational interpolation problems. These problems have a long history, associated with many classical results of Lagrange, Hermite, Prony, Padé, and other famous names. In [BSK94], [BSK99], we showed how the generalized Schur algorithm for non-Hermitian Toeplitz-like matrices [KS91], [Say92] (described in Sec. 1.11) can be used to give a recursive solution.

It is noteworthy that the solution of interpolation problems can be reduced to the determination of an appropriate fast matrix triangularization [SK92], [SKLC94], [BSK94]. This constructive view provides a nice complement to the many abstract formulations of the important topic of interpolation theory.

1.14.4 Inverse Scattering

An interesting interpretation of Schur's original recursion (1.7.16), when viewed in array form, is that it arises as the most natural way of solving the inverse scattering problem for discrete transmission lines (see [BK87b], [BK87a], [Kai87]). This interpretation gives a lot of insight into and suggests new results and new proofs for a surprisingly diverse set of problems. For example, references [BK87b], [BK87a] show how the transmission line picture gives nice interpretations of the classical Gelfand–Levitan, Marchenko, and Krein equations, and in fact yields various generalizations thereof; reference [BK87c] discusses discrete Schrödinger equations; see also [BCK88] and [RK84].

Define $\gamma_i^c = \sqrt{1-|\gamma_i|^2}$. Then the generator recursion of Schur's algorithm (cf. (1.7.13)) can be depicted graphically as a cascade of elementary sections as shown in Fig. 1.6.

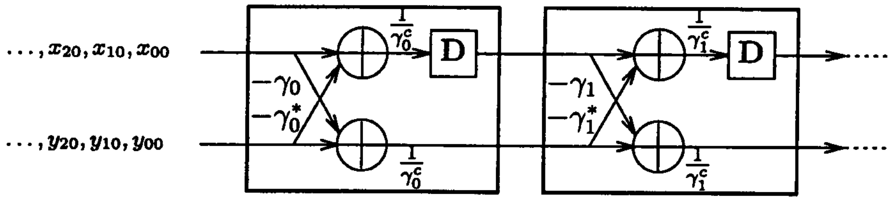

Figure 1.6. The feedforward structure (cascade network) associated with Schur's recursion.

By reversing the direction of flow in the lower line, we get a physical lossless discrete-time transmission line, as shown in Fig. 1.7, where each section is now composed of a unitary gain matrix Σ_i ($\Sigma_i\Sigma_i^* = I$) followed by a unit-time delay element,

$$\Sigma_i = \begin{bmatrix} \gamma_i^c & \gamma_i \\ -\gamma_i^* & \gamma_i^c \end{bmatrix}.$$

A physical motivation and derivation of a layered medium structure as in Fig. 1.7 can be given by showing that it corresponds to a discretization of the wave propagation (or telegrapher's) equations in an electromagnetic medium with varying local impedance; the relevant details can be found, for example, in [Kai87]. The name *reflection coefficients* for the Schur coefficients $\{\gamma_i\}$ arises from the picture in Fig. 1.7; at each section, a fraction γ_i of the incoming signal is reflected and the rest, γ_i^c, is transmitted.

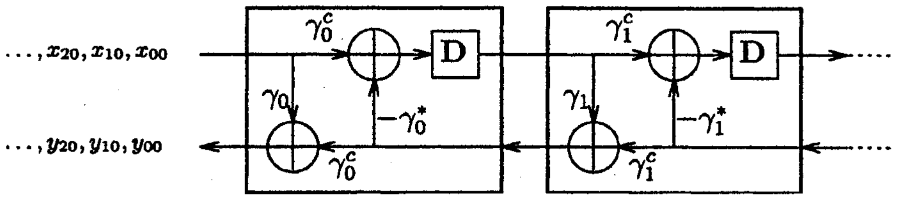

Figure 1.7. The feedback structure (transmission line) associated with Schur's recursion.

The so-called inverse-scattering problem that is associated with such layered media is the following: given an arbitrary pair of input-response sequences of a layered medium as in Fig. 1.7, say, $\{\ldots, x_{20}, x_{10}, x_{00}\}$ and $\{\ldots, y_{20}, y_{10}, y_{00}\}$, determine the medium (or reflection) parameters $\{\gamma_0, \gamma_1, \gamma_2, \ldots\}$, under the assumption that the line was initially quiescent. As mentioned above, this is a prototype of a famous problem, which has been attacked in many ways. The most widely known are methods using special choices of input sequences, based on which the inversion problem is shown to be equivalent to the solution of sets of linear equations, special forms of which are as famous as the Gelfand–Levitan, Marchenko, and Krein equations of classical inverse-scattering theory.

It turns out that a natural solution to the inverse scattering problem is Schur's array form (see [BK87b], [BK87a]). This fact leads to several useful applications in other areas including, among others, digital filter design and algebraic coding theory.