Chapter 2

STABILIZED SCHUR ALGORITHMS

Shivkumar Chandrasekaran

Ali H. Sayed

2.1 INTRODUCTION

As mentioned in Ch. 1, linear systems of equations are generally solved by resorting to the LDU factorization (or Gaussian elimination) of the coefficient matrix. But for indefinite or nonsymmetric matrices, the LDU factorization is well known to be numerically unstable if done without pivoting (see, e.g., [GV96], [Hig96], and also the discussion in Ch. 4). Moreover, since pivoting can destroy the structure of a matrix, it is not always possible to incorporate it immediately into a fast algorithm for structured matrices without potential loss of computational efficiency.

It was observed in [Hei95], however, that for Cauchy-like structured matrices, pivoting can be incorporated into fast factorization algorithms without reducing the computational efficiency of the algorithms (see the discussion in Sec. 1.13). This is because for such matrices, the displacement operators are diagonal and, therefore, column and row permutations do not destroy the Cauchy-like structure. The algorithm proposed in [Hei95] was of a hybrid type, involving Schur-type and Levinson-type operations. The technique was further used in [GKO95] to incorporate pivoting into the generalized Schur algorithm. This was achieved by first transforming different kinds of matrix structure into Cauchy-like structure and then using the so-called generator recursions of the Schur algorithm with *partial* pivoting. Sections1.13 and 4.3 of this book review this approach to factorization.

While this transformation-and-pivoting addition to the generalized Schur algorithm can be satisfactory in many situations, it still suffers from two problems. First, the procedure can 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). It also seems difficult to implement *complete* pivoting in a fast algorithm without accruing a considerable loss of efficiency. Second, the transformation into a Cauchy-like structure makes it difficult to solve a linear system of equations of a higher order by relying on the solution of a linear system of equations of a smaller order. This is because once the size of the coefficient matrix is modified, say, by appending one more row and column to it, the transformation to Cauchy-like structures has to be applied afresh to the new extended matrix and the previous calculations therefore must be repeated. In this way, one of the major features of the generalized Schur algorithm is lost, viz., the possibility to solve a sequence of nested linear systems by exploiting the results of previous calculations (as already explained in Sec. 1.13.3).

2.2 CONTRIBUTIONS AND PROBLEMS

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. In this chapter we follow such a direct route to improving and ensuring the numerical stability of the generalized Schur algorithm and, as a by-product, we shall further devise in Ch. 3 a new numerically stable solver for linear systems of equations $Rx = b$, with structured coefficient matrices R. There are different notions of numerical stability in the literature. We follow the ones suggested in [Bun85], [Bun87] and reviewed in Ch. 4. More specifically, the error bounds we present for the algorithms developed here and in Ch. 3 will be such that they guarantee (backward) numerical stability in the sense defined in Ch. 4.

This chapter and the following one provide an overview of some recent results by the authors in [CS96], [CS98] and, for this reason, some derivations are not repeated here. The main ideas and conclusions, however, are emphasized. Also, complete descriptions of the algorithms are included for ease of reference.

Our exposition highlights three contributions:

1. We first show how to modify the generalized Schur algorithm of Ch. 1 in order to guarantee a fast *and* numerically stable triangular factorization procedure for positive-definite structured matrices R. For all practical purposes, the major conclusion of this chapter is that the generalized Schur algorithm, with certain modifications, is *backward stable* for a large class of structured matrices. This conclusion extends earlier work by [BBHS95] (see also Ch. 4) on the stability of a more specialized form of the algorithm. An overview of earlier works in this direction is provided in what follows.

2. Once it is shown how to obtain a provably stable implementation of the generalized Schur algorithm for positive-definite structured matrices, we then proceed to show in Ch. 3 how the result can be used to solve in a stable manner linear systems of equations with *indefinite* and possibly *nonsymmetric* structured coefficient matrices R. In other words, we show how to use the stability results of the positive-definite case to derive stable solvers even for the indefinite and nonsymmetric cases. This is achieved by exploiting in a suitable way the embedding techniques of [KC94], which are also described in Sec. 1.8.

3. We provide a detailed numerical analysis of the proposed algorithms.

2.3 RELATED WORKS IN THE LITERATURE

As already mentioned in Sec. 1.2, one of the most frequent structures, at least in signal processing applications, is the Toeplitz structure, with constant entries along the diagonals of the matrix. A classical algorithm for the Cholesky factorization of the *inverses* of such matrices is the Levinson algorithm [Lev47], [GV96], an error analysis of which has been provided by Cybenko [Cyb80]. He showed that, in the case of positive reflection coefficients, the residual error produced by the Levinson procedure is comparable to the error produced by the Cholesky factorization [GV96, p. 191]; i.e., the algorithm is weakly stable (cf. Sec. 4.5.1).

A related analysis has been carried out by Sweet [Swe82] for the Bareiss algorithm [Bar69], which was later recognized as being closely related to the algorithm of Schur [Sch17], [Kai86]; these are fast procedures for the Cholesky factorization of the Toeplitz matrix itself rather than its inverse (cf. Ch. 1 of this book). Sweet concluded that the Bareiss algorithm is asymptotically stable.

In recent work, Bojanczyk et al. [BBHS95] further extended and strengthened the conclusions of Sweet [Swe82] by employing elementary downdating techniques [APP88], [BBDH87], [BS88] that are also characteristic of array formulations of the Schur algorithm [KS95a], [SCK95]. They considered the class of quasi-Toeplitz matrices (viz., with a generator matrix G having two columns in the definition (1.2.4)—with $p = q = 1$), which includes the Toeplitz matrix as a special case, and provided an error analysis that established that the Schur algorithm for this class of matrices is asymptotically stable.

Contributions of Our Work

The results of Bojanczyk et al. [BBHS95] motivated us to take a closer look at the numerical stability of a generalized Schur algorithm [KS95a], [Say92], [SK95a] that applies to a more general class of structured matrices, viz., all positive-definite matrices R that satisfy displacement equations of the form $R - FRF^T = GJG^T$, where F is a stable lower triangular matrix (i.e., its diagonal entries have magnitude less than unity). This class clearly includes the case of quasi-Toeplitz matrices (by choosing $F = Z$). Several complications arise in this more general case when a matrix F is used rather than Z. In this chapter we provide an overview of the results of [CS96], where we propose several enhancements to the generalized Schur algorithm of Ch. 1 in order to ensure numerical stability while evaluating the Cholesky factor $\hat{L}$ in the factorization $\hat{L}\hat{L}^T$. Hence, the current chapter is concerned with the numerical stability of the triangular factorization procedure.

Chapter 3, on the other hand, is concerned with the solution of linear systems of equations even for more general structured matrices (that need not be positive definite or even symmetric as above). More specifically, in Ch. 3 we use the stability results of the current chapter to develop a fast stable solver for linear systems of equations, $Tx = b$, with possibly indefinite or nonsymmetric structured coefficient matrices T [CS98].[5] As is well known, apart from the classical Gaussian elimination procedure, another way to solve the linear system of equations $Tx = b$ is to compute the QR factorization of the coefficient matrix T. For structured matrices, the computation has to be performed rapidly and several fast methods have been proposed earlier in the literature [BBH86], [CKL87], [Cyb83], [Cyb87], [Swe84], but none of them are numerically stable, especially since the resulting Q matrix is not guaranteed to be orthogonal (see Sec. 1.8.3 and also the discussion in Sec. 4.6).

In Ch. 3 we circumvent this difficulty by describing a new fast algorithm by the authors that provides a modified factorization for the coefficient matrix. The new algorithm relies on the observation that it is not really necessary to limit ourselves to LDU or QR factorizations of the coefficient matrix T in order to solve the linear system of equations $Tx = b$. If other factorizations for T can be obtained in a fast and stable manner, and if they are also useful for solving the linear system of equations, then these factorizations could be pursued as well. In fact, the new algorithm, rather than

[5] We are now denoting the coefficient matrix by T to distinguish it from the notation R for the R factor in the QR factorization of a matrix.

returning Q, returns two matrices Δ and Q such that Δ is triangular and the product $\Delta^{-1}Q$ is "numerically orthogonal"; it provides a factorization for the coefficient matrix T that is of the form

$$T = \Delta(\Delta^{-1}Q)R,$$

where we are now using R to denote an upper triangular matrix (just like the notation used to denote the R factor in the QR factorization of a matrix). The above factorization is in terms of three matrices $\{\Delta, Q, R\}$. The factorization is of course not unique, since we can replace Δ by any invertible matrix. The point, however, is that our algorithm returns that Δ that allows us to compensate for the fact that Q is not "numerically" orthogonal. More important, these factors are then used to solve $Tx = b$ both fast and in a backward-stable manner. The details are provided in Ch. 3.

2.3.1 Notation

In the discussion that follows we use $\|\cdot\|_2$ to denote the 2-norm of its argument (either Euclidean norm for a vector or maximum singular value for a matrix). We further assume, without loss of generality, that F is represented exactly in the computer. Also, the $\hat{\cdot}$ notation denotes computed quantities, while the $\bar{\cdot}$ notation denotes intermediate exact quantities. We further let ε denote the machine precision and n the matrix size. We also use subscripted δ's to denote quantities bounded by machine precision in magnitude and we write $O_n(\varepsilon)$ to mean $O(\varepsilon c(n))$ for some polynomial $c(n)$ in n, which we usually do not specify. The special form of $c(n)$ depends on the norm used and on other details of the implementation.

We assume that in our floating point model, additions, subtractions, multiplications, divisions, and square roots are done to high relative accuracy, i.e.,

$$fl(x \circ y) = (x \circ y)(1 + \delta),$$

where $\circ$ denotes $+$, $-$, $\times$, or $\div$ and $|\delta| \leq \varepsilon$. The same is true for the square root operation. This is true for floating point processors that adhere to the IEEE standards.

2.3.2 Brief Review of Displacement Structure

A rather detailed exposition of displacement structure can be found in Ch. 1 of this book (and also in [KS95a] for more general non-Hermitian structures). Here we highlight only some of the basic equations and notation. We shall focus in this chapter, without loss of generality, on real-valued matrices. The analysis and results can be extended to the complex case.

We start with a symmetric matrix $R \in \mathbb{R}^{n\times n}$ that satisfies a displacement equation of the form

$$R - FRF^T = GJG^T \tag{2.3.1}$$

with a "low" rank matrix G, say, $G \in \mathbb{R}^{n\times r}$ with $r \ll n$. Equation (2.3.1) uniquely defines R (i.e., it has a unique solution R) if and only if the diagonal entries of the lower triangular matrix F satisfy the condition

$$1 - f_i f_j \neq 0 \text{ for all } 0 \leq i, j \leq n - 1.$$

This uniqueness condition will be assumed throughout the chapter, although it can be relaxed in some instances—see Ch. 1 and also [KS95a].

As explained in Ch. 1, the pair (G, J) is said to be a generator pair for R since, along with F, it completely identifies R. Note, however, that while R has n^2 entries, the matrix G has nr entries and r is usually much smaller than n. Therefore, algorithms that operate on the entries of G, with the purpose of obtaining a triangular factorization for R, will generally be an order of magnitude faster than algorithms that operate on the entries of R itself. The generalized Schur algorithm is one such fast $O(rn^2)$ procedure, which receives as input data the matrices (F, G, J) and provides as output data the Cholesky factor of R.

The notion of structured matrices can also be extended to nonsymmetric matrices R. In this case, the displacement of R is generally defined with respect to two lower triangular matrices F and A (which can be the same, i.e., $F = A$—see (2.3.5)),

$$R - FRA^T = GB^T. \tag{2.3.2}$$

Again, this displacement equation uniquely defines R if and only if the diagonal entries of F and A satisfy $1 - f_i a_j \neq 0$ for all i, j, a condition that will also be met in this chapter.

Several examples of matrices with displacement structure are given in Sec. 1.3, including Toeplitz, Hankel, Pick, Cauchy, and Vandermonde matrices. The concept is perhaps best illustrated by considering the much-studied special case of a symmetric Toeplitz matrix, $T = \left[t_{|i-j|}\right]_{i,j=0}^{n-1}$, $t_0 = 1$.

Let Z denote the $n \times n$ lower triangular shift matrix with ones on the first subdiagonal and zeros elsewhere (i.e., a lower triangular Jordan block with zero eigenvalues):

$$Z \triangleq \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}. \tag{2.3.3}$$

It can be checked easily that the difference $T - ZTZ^T$ has displacement rank 2 (except when all $t_i, i \neq 0$, are zero) and a generator for T is $\{G, (1 \oplus -1)\}$, where

$$T - ZTZ^T = \begin{bmatrix} 1 & 0 \\ t_1 & t_1 \\ \vdots & \vdots \\ t_{n-1} & t_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ t_1 & t_1 \\ \vdots & \vdots \\ t_{n-1} & t_{n-1} \end{bmatrix}^T = GJG^T. \tag{2.3.4}$$

Similarly, for a nonsymmetric Toeplitz matrix $T = [t_{i-j}]_{i,j=0}^{n-1}$, we can easily verify that the difference $T - ZTZ^T$ has displacement rank 2 and that a generator (G, B) for T is

$$T - ZTZ^T = \begin{bmatrix} t_0 & 1 \\ t_1 & 0 \\ \vdots & \vdots \\ t_{n-1} & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & t_{-1} \\ \vdots & \vdots \\ 0 & t_{-n+1} \end{bmatrix}^T = GB^T. \tag{2.3.5}$$

This is a special case of (2.3.2) with $F = A = Z$. In particular, any matrix T (symmetric or not) for which $(T - ZTZ^T)$ has rank 2 is called *quasi Toeplitz*. For example, the inverse of a Toeplitz matrix is quasi Toeplitz (see Ch. 1). For higher displacement ranks, but still with $F = A = Z$, we shall say that the matrix is *shift structured*. For example, the product of two Toeplitz matrices is shift structured with displacement rank 4 (see, e.g., Ch. 1 and [KS95a]). Also, examples of structured matrices with diagonal $\{F, A\}$ in (2.3.2) are given in Ch. 1.

2.4 THE GENERALIZED SCHUR ALGORITHM

In this chapter we focus on symmetric positive-definite matrices R with displacement rank 2 with respect to a lower triangular matrix F, viz., matrices R that satisfy displacement equations of the form

$$R - FRF^T = \begin{bmatrix} u_0 & v_0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_0 & v_0 \end{bmatrix}^T, \tag{2.4.1}$$

where u_0 and v_0 denote the $n \times 1$ column vectors of G and F is lower triangular with diagonal entries whose magnitude is less than unity. The results and conclusions can be extended easily to higher displacement ranks (and will be briefly mentioned in Sec. 3.1).

Recall also from Ch. 1 that since the generator matrix G is highly nonunique, it can always be in the so-called proper form

$$G = \begin{bmatrix} x & 0 \\ x & x \\ x & x \\ \vdots & \vdots \\ x & x \end{bmatrix}. \tag{2.4.2}$$

That is, the top entry of v_1, v_{11}, can always be chosen to be zero. Indeed, assume that a generator G for R is found that does not satisfy this requirement, say,

$$G = \begin{bmatrix} u_{00} & v_{00} \\ x & x \\ \vdots & \vdots \\ x & x \end{bmatrix}.$$

It then follows from (2.4.1) that the $(0,0)$ entry of R, which is positive, is given by

$$[R]_{00} = \frac{|u_{00}|^2 - |v_{00}|^2}{1 - |f_0|^2} > 0.$$

Consequently, $|u_{00}| > |v_{00}|$ and a hyperbolic rotation Θ can always be found in order to reduce the row $\begin{bmatrix} u_{00} & v_{00} \end{bmatrix}$ to the form $\begin{bmatrix} \pm\sqrt{|u_{00}|^2 - |v_{00}|^2} & 0 \end{bmatrix}$ (see App. B). The matrix $G\Theta$ can then be used instead of G as a generator for R.

We now restate for convenience the generalized Schur algorithm in array form, which operates on the entries of (F, G, J) and provides the Cholesky factor of R. This statement is of course a special case of the algorithm derived in Sec. 1.7. We note, however, that we are now denoting the triangular factorization of the positive-definite R simply by $R = LL^T$ (rather than by $R = LD^{-1}L^T$ as in (1.6.1)).

Algorithm 2.4.1 (Generalized Schur Algorithm). *Consider a symmetric positive definite matrix $R \in \mathbb{R}^{n \times n}$ satisfying (2.4.1).*

- Input: *A stable lower triangular matrix F, a generator $G_0 = G$ in proper form, with columns denoted by u_0 and v_0, and $J = (1 \oplus -1)$.*
- Output: *The lower triangular Cholesky factor L of the unique symmetric positive-definite matrix R that satisfies (2.4.1), $R = LL^T$.*

- Computation: *Start with* (u_0, v_0), $F_0 = F$, *and repeat for* $i = 0, 1, \ldots, n-1$:

1. *Compute the matrix* $\Phi_i = (F_i - f_i I)(I - f_i F_i)^{-1}$. *Note that* Φ_i *is* $(n-i)\times(n-i)$ *lower triangular and that its first row is always zero.*

2. *Form the prearray of numbers* $\begin{bmatrix} \Phi_i u_i & v_i \end{bmatrix}$. *Since* $\begin{bmatrix} u_i & v_i \end{bmatrix}$ *is assumed in proper form and since the first entry of* $\Phi_i u_i$ *is always zero, the prearray therefore has the form*

$$\begin{bmatrix} 0 & 0 \\ \multicolumn{2}{c}{\bar{G}_{i+1}} \end{bmatrix} \triangleq \begin{bmatrix} \Phi_i u_i & v_i \end{bmatrix}.$$

That is, its top row is zero.

3. *Apply a hyperbolic rotation* Θ_i *in order to annihilate the* $(1,2)$ *entry of* $\bar{G}_{i+1}$, *thus reducing it to proper form, say,*

$$G_{i+1} \triangleq \begin{bmatrix} \times & 0 \\ \times & \times \\ \vdots & \vdots \\ \times & \times \end{bmatrix} = \bar{G}_{i+1}\Theta_i = \underbrace{\begin{bmatrix} \times & \times \\ \times & \times \\ \vdots & \vdots \\ \times & \times \end{bmatrix}}_{\bar{G}_{i+1}} \Theta_i.$$

We denote the resulting columns of G_{i+1} *by* $\{u_{i+1}, v_{i+1}\}$ *and write, more compactly,*

$$G_{i+1} = \begin{bmatrix} u_{i+1} & v_{i+1} \end{bmatrix} = \bar{G}_{i+1}\Theta_i. \tag{2.4.3}$$

4. *The ith column of the Cholesky factor* L *is given by*

$$l_i = \begin{bmatrix} 0 \\ \sqrt{1-|f_i|^2}(I - f_i F_i)^{-1} u_i \end{bmatrix}, \tag{2.4.4}$$

where the top i entries are zero.

5. F_{i+1} *is obtained by deleting the first row and column of* F_i.

◇

After n steps, the algorithm provides the Cholesky decomposition

$$R = \sum_{i=0}^{n-1} l_i l_i^T. \tag{2.4.5}$$

Recall also from Remark 2 after Alg. 1.6.2 that the successive matrices G_i that are obtained via the recursion have an interesting interpretation. Let R_i denote the Schur complement of R with respect to its leading $i \times i$ submatrix. That is, $R_0 = R$, R_1 is the Schur complement with respect to the $(0,0)$th top left entry of R, R_2 is the Schur complement with respect to the 2×2 top left submatrix of R, and so on. The matrix R_i is therefore $(n-i) \times (n-i)$. Then

$$R_i - F_i R_i F_i^T = G_i J G_i^T. \tag{2.4.6}$$

In other words, G_i is a generator matrix for the ith Schur complement, which is also structured. Note that both G_i and $\bar{G}_i$ can be regarded as generator matrices for the ith Schur complement R_i.

2.5 A LIMIT TO NUMERICAL ACCURACY

Given a symmetric positive-definite matrix R (not necessarily structured), if its Cholesky factor is evaluated by any standard backward stable method that operates on the entries of R, e.g., by Gaussian elimination (see [GV96, Ch. 4] and also Ch. 1) the corresponding error bound is given by

$$\|R - \hat{L}\hat{L}^T\|_2 \leq O_n(\varepsilon)\|R\|_2,$$

where ε is the machine precision.

A fundamental question that needs to be answered then is the following: given (F, G, J) but not R, how accurately can we expect to be able to compute the Cholesky factorization of R *irrespective* of the algorithm used (*slow or fast*)? The example and discussion that follows justifies the following conclusion [CS96].

Lemma 2.5.1 (Limit of Accuracy). *Irrespective of the algorithm we use (*slow or fast*), if the input data is (F, G, J), for a general lower triangular F, we cannot expect a better bound than*

$$\|R - \hat{L}\hat{L}^T\|_2 \leq O_n(\varepsilon)\ \|(I - F \otimes F)^{-1}\|_2\ \|u_0\|_2^2. \tag{2.5.1}$$

Proof: The claim is established in [CS96] by constructing a simple example. We highlight the main steps here.

To begin with, note that just representing (F, G) in finite precision already induces round-off errors. This fact in turn imposes limits on how accurate an algorithm that employs (F, G) can be.

Consider the following example. Let F be a stable diagonal matrix with distinct entries $\{f_i\}$ and assume f_0 is the largest in magnitude. Let the entries of the column vectors u_0 and v_0 be constructed as follows:

$$u_{i0} = \left(\frac{1}{2}\right)^{i-1}, \quad v_{i0} = \gamma f_i u_{i0}, \quad i \geq 1,$$

where γ is chosen such that $0 < \gamma < 1$. The unique matrix R that solves (2.4.1) for the given (F, u_0, v_0) can be shown to be symmetric positive definite.

When the data (u_0, v_0) are stored in finite precision, round-off errors are bound to occur. Let us assume that only a relative perturbation occurs in the first entry of u_0, while all other entries of u_0 and v_0 remain unchanged. That is, let us define the perturbed vectors $\hat{u}_0$ and $\hat{v}_0$ with

$$\hat{u}_{00} = u_{00}(1 + \delta), \quad \hat{u}_{i0} = u_{i0} \quad i \geq 1, \quad \hat{v}_0 = v_0,$$

where δ is a small number (for example, for round-off errors, $|\delta|$ is smaller than machine precision). We also assume that F is stored exactly. Hence, the only source of error we are assuming is in u_{00}.

An algorithm that is intended to factor R will in fact be factoring the matrix $\hat{R}$ that is defined as the unique solution of the following displacement equation with the perturbed generator matrix,

$$\hat{R} - F\hat{R}F^T = \hat{G}J\hat{G}^T, \quad \hat{G} = \begin{bmatrix} \hat{u}_0 & \hat{v}_0 \end{bmatrix}.$$

The difference between this matrix and the original matrix R is denoted by the error matrix $E = R - \hat{R}$. How big can E be? It is easy to verify that E is the unique solution of

$$E - FEF^T = GJG^T - \hat{G}J\hat{G}^T = u_0 u_0^T - \hat{u}_0\hat{u}_0^T.$$

Using this fact, it can be verified that [CS96]

$$\|E\|_2 \geq \frac{3}{4}|2\delta+\delta^2|\;\|(I-F\otimes F)^{-1}\|_2\;\|u_0\|_2^2,$$

where $\otimes$ denotes the Kronecker product. This expression provides a *lower bound* on the norm of the error matrix. It is further argued in [CS96] that by choosing f_0 and γ sufficiently close to one, the norm of the original matrix R can be made much smaller than the above bound.

Hence, in general, we cannot expect the error norm, $\|R-\hat{L}\hat{L}^T\|$, for any algorithm (slow or fast) that uses (F,G,J) as input data (but not R) to be as small as $c_0|\delta|\|R\|$ for some constant c_0.

◊

The above perturbation analysis indicates the best accuracy that can be expected from *any* finite precision algorithm that uses the generator matrix as the input data. The issue now is to show that the generalized Schur algorithm can essentially achieve this bound if certain care is taken during its implementation. We shall show in this chapter that, in general, this requires that we incorporate four kinds of enhancement:

1. A careful implementation of the hyperbolic rotation Θ_i that is needed in each step of the algorithm (Sec. 2.6).

2. A careful implementation of the Blaschke-vector product $\Phi_i u_i$ that is also needed in each step of the algorithm (Sec. 2.7).

3. Enforcing positive definiteness of the successive Schur complements to avoid breakdown (Sec. 2.8).

4. Control of potential growth of the norms of the successive generator matrices (Sec. 2.11). We may remark that pivoting strategies can be useful in controlling generator growth when F is diagonal with norm close to unity.

For all practical purposes, the major conclusion (see Sec. 2.9) of the analysis will be that the modified Schur algorithm is a backward-stable procedure for a large class of positive-definite structured matrices.

2.6 IMPLEMENTATIONS OF THE HYPERBOLIC ROTATION

Each step (2.4.3) of the generalized Schur algorithm requires the application of a hyperbolic rotation Θ_i. The purpose of the rotation is to rotate the top row of the $\bar{G}_{i+1}$, which is the second row of the $\begin{bmatrix} \Phi_i u_i & v_i \end{bmatrix}$, to proper form. If we denote the top row of $\bar{G}_{i+1}$ by

$$\begin{bmatrix} (\Phi_i u_i)_1 & v_{i1} \end{bmatrix} \stackrel{\Delta}{=} \begin{bmatrix} \alpha_i & \beta_i \end{bmatrix},$$

then the expression for a hyperbolic rotation that transforms it to the form

$$\begin{bmatrix} \pm\sqrt{|\alpha_i|^2-|\beta_i|^2} & 0 \end{bmatrix}$$

is given by

$$\Theta_i = \frac{1}{\sqrt{1-\rho_i^2}}\begin{bmatrix} 1 & -\rho_i \\ -\rho_i & 1 \end{bmatrix}, \quad \text{where } \rho_i = \frac{\beta_i}{\alpha_i}. \tag{2.6.1}$$

The positive definiteness of R guarantees $|\rho_i| < 1$.

Expression (2.4.3) shows that in infinite precision, the generator matrices G_{i+1} and $\bar{G}_{i+1}$ must satisfy the fundamental requirement

$$G_{i+1}JG_{i+1}^T = \bar{G}_{i+1}J\bar{G}_{i+1}^T. \tag{2.6.2}$$

Obviously, this condition cannot be guaranteed in finite precision. But it turns out that with an appropriate implementation of the transformation (2.4.3), equality (2.6.2) can be guaranteed to within a "small" error. The need to enforce the condition in finite precision was first observed for the $F = Z$ case by Bojanczyk et al. [BBHS95].

2.6.1 Direct Implementation

A naive implementation of the hyperbolic transformation (2.4.3) can lead to large errors. Indeed, in finite precision, if we apply Θ_i directly to $\bar{G}_{i+1}$ we obtain a computed matrix $\widehat{G}_{i+1}$ such that [CS96]

$$\|\widehat{G}_{i+1}J\widehat{G}_{i+1}^T - \bar{G}_{i+1}J\bar{G}_{i+1}^T\|_2 \leq O_n(\varepsilon)\ \|\bar{G}_{i+1}\|_2^2\ \|\Theta_i\|_2^2. \tag{2.6.3}$$

But since $\|\Theta_i\|$ can be large, the computed quantities are not guaranteed to satisfy relation (2.6.2) to sufficient accuracy. This possibly explains the disrepute into which fast algorithms have fallen.

Interestingly though, Bojanczyk et al. [BBHS95] showed that for the special case $F = Z$ and displacement rank $r = 2$, the direct implementation of the hyperbolic rotation still leads to an asymptotically backward stable algorithm. This conclusion, however, does not hold for higher displacement ranks. Stewart and Van Dooren [SD97b] showed that for $F = Z$ and $r > 2$, the direct implementation of the hyperbolic rotation can be unstable.

We proceed to review alternative methods for implementing the hyperbolic rotation that can be used for general F, including $F = Z$, and also for higher displacement ranks.

2.6.2 Mixed Downdating

One possible way to ameliorate the above problem is to employ the mixed downdating procedure as suggested by Bojanczyk et al. [BBHS95], [BBDH87].

Assume we apply a hyperbolic rotation Θ to a row vector $\begin{bmatrix} x & y \end{bmatrix}$, say,

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \frac{1}{\sqrt{1-|\rho|^2}} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}. \tag{2.6.4}$$

Then, more explicitly,

$$x_1 = \frac{1}{\sqrt{1-|\rho|^2}}\left[x - \rho y\right], \tag{2.6.5}$$

$$y_1 = \frac{1}{\sqrt{1-|\rho|^2}}\left[-\rho x + y\right]. \tag{2.6.6}$$

Solving for x in terms of x_1 from the first equation and substituting into the second equation we obtain

$$y_1 = -\rho x_1 + \sqrt{1-|\rho|^2}\, y. \tag{2.6.7}$$

An implementation that is based on (2.6.5) and (2.6.7) is said to be in mixed downdating form. It has better numerical stability properties than a direct implementation of Θ as in (2.6.4).

In the above mixed form, we first evaluate x_1 and then use it to compute y_1. We can obtain a similar procedure that first evaluates y_1 and then uses it to compute x_1. For this purpose, we solve for y in terms of y_1 from (2.6.6) and substitute into (2.6.5) to obtain

$$x_1 = -\rho y_1 + \sqrt{1-|\rho|^2}\,x\,. \tag{2.6.8}$$

Equations (2.6.6) and (2.6.8) represent the second mixed form.

Using this scheme to implement the hyperbolic transformation (2.4.3) guarantees (cf. [BBHS95], [BBDH87])

$$\|\widehat{G}_{i+1}J\widehat{G}_{i+1}^T - \bar{G}_{i+1}J\bar{G}_{i+1}^T\|_2 \le O_n(\varepsilon)\left(\|\bar{G}_{i+1}\|_2^2 + \|\widehat{G}_{i+1}\|_2^2\right).$$

This bound is sufficient, when combined with other modifications suggested in Secs. 2.7 and 2.11, to make the algorithm numerically reliable (Sec. 2.9).

2.6.3 The Orthogonal-Diagonal Procedure

An alternative scheme that was employed in [CS96] is based on using the singular value decomposition (SVD) representation of a hyperbolic rotation Θ. Its good numerical properties derive from the fact that the hyperbolic rotation is applied as a sequence of orthogonal and diagonal matrices, which we shall refer to as the orthogonal-diagonal (OD) procedure. Its other advantage is that it is a general technique that can be applied in other situations. It can be implemented with the same operation count as the mixed downdating algorithm of [BBHS95].

It is straightforward to verify that any hyperbolic rotation of the form (2.6.1) admits the following eigen(SVD-)decomposition:

$$\Theta_i = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} \sqrt{\frac{\alpha_i+\beta_i}{\alpha_i-\beta_i}} & 0 \\ 0 & \sqrt{\frac{\alpha_i-\beta_i}{\alpha_i+\beta_i}} \end{bmatrix}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\frac{1}{\sqrt{2}} = Q_i D_i Q_i^T,$$

where the matrix

$$Q_i = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

is orthogonal ($Q_i Q_i^T = I$).

If the eigendecomposition $Q_i D_i Q_i^T$ is now applied to the prearray $\bar{G}_{i+1}$ in (2.4.3), then it can be shown that the computed generator matrix $\widehat{G}_{i+1}$ satisfies [CS96]

$$(\widehat{G}_{i+1} + E_{2,i+1}) = (\bar{G}_{i+1} + E_{1,i+1})\Theta_i \tag{2.6.9}$$

and

$$\|\widehat{G}_{i+1}J\widehat{G}_{i+1}^T - \bar{G}_{i+1}J\bar{G}_{i+1}^T\|_2 \le O_n(\varepsilon)\left(\|\bar{G}_{i+1}\|_2^2 + \|\widehat{G}_{i+1}\|_2^2\right)$$

with

$$\|E_{1,i+1}\|_2 \le O_n(\varepsilon)\,\|\bar{G}_{i+1}\|_2, \quad \|E_{2,i+1}\|_2 \le O_n(\varepsilon)\,\|\widehat{G}_{i+1}\|_2.$$

Algorithm 2.6.1 (The OD Procedure). *Given a hyperbolic rotation Θ with reflection coefficient $\rho = \beta/\alpha$, $|\rho| < 1$, and a prearray row vector $\begin{bmatrix} x & y \end{bmatrix}$, the postarray row vector $\begin{bmatrix} x_1 & y_1 \end{bmatrix}$ can be computed as follows:*

$$\begin{aligned}
\begin{bmatrix} x' & y' \end{bmatrix} &= \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \\
\begin{bmatrix} x'' & y'' \end{bmatrix} &= \begin{bmatrix} x' & y' \end{bmatrix} \begin{bmatrix} \frac{1}{2}\sqrt{\frac{\alpha+\beta}{\alpha-\beta}} & 0 \\ 0 & \frac{1}{2}\sqrt{\frac{\alpha-\beta}{\alpha+\beta}} \end{bmatrix}, \\
\begin{bmatrix} x_1 & y_1 \end{bmatrix} &= \begin{bmatrix} x'' & y'' \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}.
\end{aligned}$$

◇

The above algorithm guarantees the error bounds

$$\begin{bmatrix} \hat{x}_1 + e_1 & \hat{y}_1 + e_2 \end{bmatrix} = \begin{bmatrix} x + e_3 & y + e_4 \end{bmatrix} \Theta$$

with

$$\left\| \begin{bmatrix} e_1 & e_2 \end{bmatrix} \right\|_2 \le O_n(\varepsilon) \left\| \begin{bmatrix} \hat{x}_1 & \hat{y}_1 \end{bmatrix} \right\|_2, \quad \left\| \begin{bmatrix} e_3 & e_4 \end{bmatrix} \right\|_2 \le O_n(\varepsilon) \left\| \begin{bmatrix} x & y \end{bmatrix} \right\|_2.$$

2.6.4 The H Procedure

Another method introduced in [CS96] is the following. Let $\rho = \beta/\alpha$ be the reflection coefficient of a hyperbolic rotation Θ,

$$\Theta = \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix},$$

with $|\rho| < 1$. Let $\begin{bmatrix} x_1 & y_1 \end{bmatrix}$ and $\begin{bmatrix} x & y \end{bmatrix}$ be the postarray and prearray rows, respectively:

$$\begin{bmatrix} x_1 & y_1 \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \Theta, \text{ with } |x| > |y|.$$

Algorithm 2.6.2 (The H Procedure). *Given a hyperbolic rotation Θ with reflection coefficient $\rho = \beta/\alpha$, $|\rho| < 1$, and a prearray $\begin{bmatrix} x & y \end{bmatrix}$ with $|x| > |y|$, the postarray $\begin{bmatrix} x_1 & y_1 \end{bmatrix}$ can be computed as follows:*

```
If $\frac{\beta}{\alpha}\frac{y}{x} < 1/2$
  then $\xi = 1 - \frac{\beta}{\alpha}\frac{y}{x}$
else
  $d_1 = \frac{|\alpha|-|\beta|}{|\alpha|}$, $d_2 = \frac{|x|-|y|}{|x|}$
  $\xi = d_1 + d_2 - d_1 d_2$
endif
$x_1 = \frac{|\alpha| x \xi}{\sqrt{(\alpha-\beta)(\alpha+\beta)}}$
$y_1 = x_1 - \sqrt{\frac{\alpha+\beta}{\alpha-\beta}}\,(x - y)$.
```

◇

The advantage of the H procedure is that the computed quantities $\widehat{x}_1$ and $\widehat{y}_1$ will satisfy the equation

$$\begin{bmatrix} \widehat{x}_1 + e_1' & \widehat{y}_1 + e_2' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \Theta, \tag{2.6.10}$$

where the error terms satisfy

$$|e_1'| \leq O_n(\varepsilon)|\widehat{x}_1|, \quad |e_2'| \leq O_n(\varepsilon)(|\widehat{x}_1| + |\widehat{y}_1|). \tag{2.6.11}$$

Comparing with (2.6.9) we see that the prearray is not perturbed. Moreover, we shall show in Sec. 2.13.2 that by a slight modification we can further enforce that $|\widehat{x}_1| > |\widehat{y}_1|$, which is needed to prevent breakdown in the algorithm. (If $|x| < |y|$, then it can be seen that $\begin{bmatrix} y & x \end{bmatrix} \Theta = \begin{bmatrix} y_1 & x_1 \end{bmatrix}$. Therefore, without loss of generality, we considered above only the case $|x| > |y|$.)

The the H procedure requires $5n$ to $7n$ multiplications and $3n$ to $5n$ additions. It is therefore costlier than the OD procedure, which requires $2n$ multiplications and $4n$ additions. But the H procedure is forward stable (cf. (2.6.10)), whereas the OD method is only stable (cf. (2.6.9)).

From now on we shall denote by $\widehat{u}_{i+1}$ and $\widehat{v}_{i+1}$ the computed generator columns at step i, i.e., $\widehat{G}_{i+1} = \begin{bmatrix} \widehat{u}_{i+1} & \widehat{v}_{i+1} \end{bmatrix}$, starting with $\bar{G}_i = \begin{bmatrix} \Phi_i \widehat{u}_i & \widehat{v}_i \end{bmatrix}$.

2.7 IMPLEMENTATION OF THE BLASCHKE MATRIX

Each step of the Schur algorithm also requires multiplying the Blaschke matrix Φ_i by u_i. (Recall that the top entry of $\Phi_i u_i$ is always zero and, hence, can be ignored in the computation.) In this section, we consider the following two cases:

- F is stable and diagonal, in which case Φ_i itself is diagonal and its entries are given by
$$\left\{ \frac{f_j - f_i}{1 - f_i f_j} \right\}.$$
- F is strictly lower triangular, e.g., $F = Z$, $F = (Z \oplus Z)$, or other more involved choices. In these situations, the matrix Φ_i is equal to F_i itself since the f_i are all zero, $\Phi_i = F_i$.

2.7.1 The Case of Diagonal F

The goal of this section is to show how to compute $\Phi_i \widehat{u}_i$ to high componentwise relative accuracy (i.e., high relative accuracy for each component of the computed vector). Here, $\widehat{u}_i$ denotes the computed value of u_i.

The numerator of each diagonal entry of Φ_i can be computed to high relative accuracy as

$$fl(f_j - f_i) = (f_j - f_i)(1 + \delta_1).$$

Computing the denominator $x_{ij} \overset{\Delta}{=} (1 - f_i f_j)$ to high relative accuracy is a bit trickier, as the following example shows.

Let $f_1 = f_2 = 0.998842$. Then in 6-digit arithmetic $1 - f_1 f_2 \approx 2.31500 \times 10^{-3}$, whereas the actual answer is $2.31465903600 \times 10^{-3}$. Therefore, the relative error is approximately 1.5×10^{-4}. Using the scheme given below, we find $1 - f_1 f_2 \approx 2.31466 \times 10^{-3}$. The relative error is now approximately 4.2×10^{-7}.

The scheme we use to compute x_{ij} is as follows:

```
If $f_i f_j < 1/2$
  then $x_{ij} = 1 - f_i f_j$
else
  $d_j = 1 - |f_j|$,  $d_i = 1 - |f_i|$,
  $x_{ij} = d_i + d_j - d_i d_j$.
```

It can be shown that this scheme ensures that x_{ij} is computed to high relative accuracy [CS96], viz., that

$$\hat{x}_{ij} = x_{ij}\,(1 + 33\delta_{11})\,.$$

Knowing how to compute Φ_i to componentwise accuracy, and since Φ_i is diagonal, the entries of $\Phi_i \hat{u}_i$ can be computed to componentwise high relative accuracy. More specifically,

$$fl\,(\Phi_i \hat{u}_i)_k = (\Phi_i \hat{u}_i)_k (1 + 72\delta_{12}). \tag{2.7.1}$$

We should remark that this scheme is not totally successful when F is a general triangular matrix (for example, when F is bidiagonal). A way around this difficulty will be addressed elsewhere. But for a strictly lower triangular F, the situation is far simpler as shown in the next subsection.

For now, let us consider how to compute the nonzero part of the l_i's. Define

$$\bar{l}_i \triangleq \begin{bmatrix} 0 \\ \sqrt{1 - f_i^2}\,(I - f_i F)^{-1} \hat{u}_i \end{bmatrix} \tag{2.7.2}$$

with i leading zeros. We use the expression

$$\frac{\sqrt{(1 - f_i)(1 + f_i)}}{1 - f_i f_j} (\hat{u}_i)_k$$

to evaluate the nonzero entries of $\bar{l}_i$, with the technique explained above for the denominator $(1 - f_i f_j)$. (The notation $(\hat{u}_i)_k$ denotes the kth entry of $\hat{u}_i$.) Then we can also show that the nonzero entries of $\bar{l}_i$ and $\hat{l}_i$ satisfy

$$(\hat{l}_i)_k = (\bar{l}_i)_k (1 + c_1(n)\delta_{13}) \tag{2.7.3}$$

for some polynomial in n, $c_1(n)$.

2.7.2 The Case of Strictly Lower Triangular F

When F is strictly lower triangular we obtain $\Phi_i = F_i$. Hence, here we use the standard matrix-vector multiplication to evaluate $\Phi_i u_i$ and the computed quantities will then satisfy [GV96]

$$\|fl(F_i \hat{u}_i) - F_i \hat{u}_i\|_2 \leq O_n(\varepsilon) \|F\|_2 \; \|\hat{u}_i\|_2.$$

Moreover, since $f_i = 0$,

$$\bar{l}_i = \begin{bmatrix} 0 \\ \hat{u}_i \end{bmatrix} = \hat{l}_i. \tag{2.7.4}$$

2.8 ENFORCING POSITIVE DEFINITENESS

The computed successive Schur complements have to be positive definite to guarantee that the successive reflection coefficients are all strictly less than unity in magnitude. These facts can be enforced in finite precision by imposing a condition on the smallest eigenvalue of R and by introducing a computational enhancement during the evaluation of the reflection coefficients.

Define the matrix S_i that solves the displacement equation

$$S_i - F_i S_i F_i^T = \widehat{u}_i \widehat{u}_i^T - \widehat{v}_i \widehat{v}_i^T. \tag{2.8.1}$$

If the $\{\widehat{u}_i, \widehat{v}_i\}$ were exact and equal to $\{u_i, v_i\}$, then the S_i would correspond to the ith Schur complement R_i of R (cf. (2.4.6)). The matrix R_i is necessarily positive definite since R is positive definite. However, because we are now dealing with computed values rather than exact values, we need to guarantee that the computed generator columns $\{\widehat{u}_i, \widehat{v}_i\}$ define a positive-definite matrix S_i.

It was shown in [CS96] that this can be guaranteed by imposing a condition on the smallest eigenvalue of R, viz., by guaranteeing that R is sufficiently positive definite. Define

$$\bar{R} = \sum_{i=0}^{n-1} \bar{l}_i \bar{l}_i^T,$$

where the $\bar{l}_i$ are intermediate exact quantities computed via (2.7.2). Then we can ensure the positive definiteness of the computed Schur complements if

$$\lambda_{\min}(R) \;>\; O_n(\varepsilon)\|(I - F \otimes F)^{-1}\|_2 \,(2 + \|F\|_2^2) \left[\|\bar{R}\|_2 + \sum_{j=0}^{n-1} \|\widehat{u}_j\|_2^2 \right] . \tag{2.8.2}$$

The condition further guarantees, due to positive definiteness, that $|\Phi_i \widehat{u}_i|_2 > |\widehat{v}_i|_2$. (The inequality compares the second entries of $\Phi_i \widehat{u}_i$ and $\widehat{v}_i$.) This ensures that the reflection coefficients will be smaller than one in magnitude, a condition that we can enforce in finite precision as follows:

```
If $|fl(\Phi_i \widehat{u}_i)|_2 < |\widehat{v}_i|_2$ then
    set $fl(\Phi_i \widehat{u}_i)_2 = |\widehat{v}_i|_2 (1 + 3\varepsilon)\mathrm{sign}(fl(\Phi_i \widehat{u}_i)_2)$.
```

This enhancement, along with condition (2.8.2), can be shown to guarantee that the algorithm will be completed without any breakdowns.

2.9 ACCURACY OF THE FACTORIZATION

Using the above enhancements (regarding the implementations of the hyperbolic rotation and the Blaschke-vector product and the enforcement of positivity), it was shown in [CS96] that the following error bound holds.

Theorem 2.9.1 (Error Bound). *Consider a symmetric positive-definite matrix* $R \in \mathbb{R}^{n \times n}$ *satisfying* (2.4.1) *and* (2.8.2), *with a stable diagonal or strictly lower triangular* F.

The generalized Schur algorithm, when implemented as detailed above (see also listing in App. 2.A*), guarantees the following error bound:*

$$\left\|R-\sum_{i=0}^{n-1}\hat{l}_i\hat{l}_i^T\right\|_2 \le O_n(\varepsilon)\|(I-F\otimes F)^{-1}\|_2\,(2+\|F\|_2^2)\left[\|\bar{R}\|_2+\sum_{j=0}^{n-1}\|\hat{u}_j\|_2^2\right]. \qquad (2.9.1)$$

◇

The term $\|(I-F\otimes F)^{-1}\|_2$ in the error bound is expected from the perturbation analysis of Sec. 2.5. However, the presence of the norms of the successive generators makes the error bound larger than the bound suggested by the perturbation analysis, which depends only on the norm of the first generator matrix.

The natural question then is, How big can the norm of the generators be? We consider the following cases separately.

1. ***F* strictly lower triangular.** Using (2.4.4) we can verify that $\|u_i\|_2^2 \le \|R\|_2$. It then follows that the error bound is as good as can be expected from the perturbation analysis of Sec. 2.5.

2. ***F* strictly lower triangular and contractive.** More can be said if F is further assumed to be contractive ($\|F\|_2 \le 1$). To see this, we first note that the error bound in the case when F is strictly lower triangular can be rewritten as

$$\left\|R-\sum_{i=0}^{n-1}\hat{l}_i\hat{l}_i^T\right\|_2 \le O_n(\varepsilon)(2+\|F\|_2^2)\left(\sum_{i=1}^{n}\|F^i\|_2^2\right)\left[\|\bar{R}\|_2+\left(\sum_{i=0}^{n-1}\|\hat{u}_i\|_2^2\right)\right].$$

When F is contractive, this further implies that

$$\left\|R-\sum_{i=0}^{n-1}\hat{l}_i\hat{l}_i^T\right\|_2 \le O_n(\varepsilon)\left[\|\bar{R}\|_2+\left(\sum_{i=0}^{n-1}\|\hat{u}_i\|_2^2\right)\right].$$

But since we also have $\|u_i\|_2^2 \le \|R\|_2$, we conclude that the factorization algorithm is backward stable.

This result applies to the important class of positive-definite quasi-Toeplitz matrices, which corresponds to $F = Z$. In this case, the above conclusion strengthens the result of Bojanczyk et al. [BBHS95], which states that for quasi-Toeplitz symmetric positive-definite matrices, the Schur algorithm is asymptotically backward stable. Our analysis shows that the modified algorithm proposed here is backward stable provided the smallest eigenvalue of the quasi-Toeplitz matrix satisfies

$$\lambda_{\min}(R) > O_n(\varepsilon)\left[\|\bar{R}\|_2+\sum_{j=0}^{n-1}\|\hat{u}_j\|_2^2\right].$$

If F is strictly lower triangular but noncontractive, then the error norm can possibly depend on $\|(I-F\otimes F)^{-1}\|_2$, as shown in the previous case.

3. ***F* diagonal.** Using the fact that R is positive definite and (2.4.4), we can verify that

$$\|u_i\|_2^2 \le \|(I-F\otimes F)^{-1}\|_2\,(1+\|F\|^2)^2\,\|R\|_2. \qquad (2.9.2)$$

This shows that the growth of the generators depends on $\|(I - F \otimes F)^{-1}\|_2$. This may suggest that the norm of the error can become very large when the magnitude of the diagonal entries of F becomes close to one. But this is not necessarily the case (see also the numerical example in the next section) since we can further strengthen the error bound as follows.

Define

$$\rho_{ji} = \frac{\widehat{v}_{ji}}{\widehat{u}_{ji}}.$$

It then holds in the diagonal case that

$$\left\| R - \sum_{i=0}^{n-1} \widehat{l}_i \widehat{l}_i^T \right\|_2 \leq O_n(\varepsilon) \frac{\sum_{i=1}^{n} \|\bar{R}_i\|_2}{1 - \max_{j,i}\left(\rho_{ji}^2\right)}, \tag{2.9.3}$$

where the $\bar{R}_i$ are defined by

$$\bar{R}_i = \sum_{j=0}^{i-1} \bar{l}_i \bar{l}_i^T + \begin{bmatrix} 0 & 0 \\ 0 & S_i \end{bmatrix}.$$

The bound (2.9.3) is independent of the $\{f_i\}$! In other words, if the coefficients $\rho_{j,i}$ defined above are sufficiently smaller than one, then the algorithm will still be backward stable irrespective of how close the $\{|f_i|\}$ are to one.

What does this say about the stability of the generalized Schur algorithm for a diagonal and stable F? Clearly, when the eigenvalues of F are sufficiently far from one the method has excellent numerical stability. The algorithm degrades as the eigenvalues of F get closer to one. This is to be expected from the perturbation analysis (whether we use a slow or a fast algorithm). However, if the generators grow rapidly (i.e., as fast as (2.9.2)), then the algorithm degrades faster than the rate predicted by the perturbation analysis.

Is there anything further we can do to ameliorate this problem? One thing we have not considered yet is pivoting, which is possible only when F is diagonal.

2.10 PIVOTING WITH DIAGONAL F

When F is diagonal it is possible to accommodate pivoting into the algorithm [Hei95]; it corresponds to reordering the f_j's, u_{ji}'s, and v_{ji}'s identically at the ith iteration of the algorithm. This has the effect of computing the Cholesky factorization of PRP^T, where P is the product of all the permutations that were carried out during the algorithm.

In finite precision, pivoting strategies are employed in classical Cholesky factorization algorithms when the positive-definite matrix is numerically singular. *In the context of the generalized Schur algorithm, the main motivation for pivoting should be to keep the norm of the generator matrices as small as possible* (see also Sec. 4.4.2). This is suggested by the expression for the error bound in (2.9.1), which depends on the norm of the generators. Note that this motivation has little to do with the size of the smallest eigenvalue of the matrix.

We would like to emphasize that pivoting is necessary only when the norm of F is very close to one, as otherwise the generators do not grow appreciably (cf. (2.9.2)).

2.10.1 A Numerical Example

The first question that arises then is whether there exists a pivoting strategy that guarantees a small growth in the norm of the generators. Unfortunately, we have numerical examples that show that irrespective of which pivoting strategy is employed, the norms of the generators may not exhibit significant reduction.

Consider the matrix R that satisfies the displacement equation $R - FRF^T = GJG^T$ with

$$G = \begin{bmatrix} 0.26782811166721 & 0.26782805810159 \\ 0.65586390188981 & -0.65586311485320 \\ 0.65268528182561 & 0.65268365011256 \\ 0.26853783287812 & -0.26853149538590 \end{bmatrix}, \quad J = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

and

$$F = \text{diagonal}\{0.9999999, -0.9999989, 0.9999976, -0.9999765\}.$$

The matrix R is positive definite. Table 2.1 lists the values of the sum $\sum_{j=0}^{n-1} \|\hat{u}_j\|_2^2$ (scaled by 10^{-6} and denoted by S) for all the 24 possible pivoting options of the rows of the generator matrix G. The results indicate that none of the pivoting options significantly reduces the size of the generators.

2.10.2 The Case of Positive F

This raises the next question: Is pivoting useful at all? It is useful when the entries of the F matrix are strictly positive (or negative). In this case, we permute the entries of F (and, correspondingly, the entries of u_1 and v_1) such that the diagonal of F is in increasing order in magnitude. Then it can be shown that [CS96]

$$\|\hat{u}_i\|_2 \leq \|\bar{R}\|_2^{1/2},$$

which makes the first-order term of the upper bound on E depend only on the first power of $\|(I - F \otimes F)^{-1}\|_2$.

Table 2.1. Values of the scaled sum S for all 24 pivoting options.

1.	5.30	9.	0.41	17.	0.83
2.	5.30	10.	0.41	18.	0.83
3.	5.21	11.	0.40	19.	0.83
4.	5.21	12.	0.40	20.	0.83
5.	5.03	13.	0.40	21.	0.04
6.	5.03	14.	0.40	22.	0.04
7.	0.83	15.	0.04	23.	0.04
8.	0.83	16.	0.04	24.	0.04

2.10.3 The Nonpositive Case

When F is not positive, the example in Sec. 2.10.1 suggests that pivoting may not help in general. However, it may still be beneficial to try a heuristic pivoting strategy to control the growth of the generators. Ideally, at the ith iteration we should pick the row of the prearray $\bar{G}_{i+1}$ which would lead to the smallest (in norm) postarray G_{i+1}. Since there seems to be no efficient way to do this we suggest picking the row that leads to the smallest reflection coefficient (in magnitude) for the hyperbolic rotation Θ_i. As suggested by the example of Sec. 2.10.1, an alternate strategy would be to order the f_i's in increasing order of magnitude.

We may stress that pivoting is relevant *only* when the norm of F is very close to one, as indicated by the error bound (2.9.1) and by (2.9.2).

2.11 CONTROLLING THE GENERATOR GROWTH

We have shown before that the generators do not grow (i) if F is strictly lower triangular and contractive or (ii) if F is a positive diagonal matrix with increasing diagonal entries. We now show how to control the generator growth in general using an idea suggested by Gu [Gu95a].

It follows from (2.8.1) that

$$\|\widehat{G}_i J \widehat{G}_i^T\|_2 = \|S_i - F_i S_i F_i^T\|_2.$$

Let $W_i \Lambda_i W_i^T$ denote the eigendecomposition of $\widehat{G}_i J \widehat{G}_i^T$, where Λ_i is a 2×2 real diagonal matrix with $(\Lambda_i)_{11} > 0$ and $(\Lambda_i)_{22} < 0$. Then $W_i\sqrt{|\Lambda_i|}$ can be taken as a generator for S_i with the desirable property that

$$\left\|W_i\sqrt{|\Lambda_i|}\right\|_2^2 = \|\Lambda_i\|_2 = \|S_i - F_i S_i F_i^T\|_2 \le \|S_i\|_2 \left(1 + \|F\|_2^2\right) \le \|\bar{R}_i\|_2 \left(1 + \|F\|_2^2\right),$$

where $\|\bar{R}_i\|_2 \approx \|R\|_2$, to first order in ε.

Therefore, whenever the generator grows, i.e., $\|\widehat{G}_i\|_2^2$ becomes larger than a given threshold (say, $2\|R\|_2 \left(1 + \|F\|_2^2\right)$), we can replace it by $W_i\sqrt{|\Lambda_i|}$. This computation can be done in $O((n-i)r^2 + r^3)$ flops ($r = 2$ in the case under consideration) by first computing the QR factorization of $\widehat{G}_i$, say,

$$\widehat{G}_i = Q_i P_i, \quad Q_i Q_i^T = I,$$

and then computing the eigendecomposition of the 2×2 matrix $P_i J P_i^T$. We can then get W_i by multiplying Q_i by the orthogonal eigenvector matrix of $P_i J P_i^T$.

2.12 SOLUTION OF LINEAR SYSTEMS OF EQUATIONS

The analysis in the earlier sections suggests that for $\|F\|_2$ sufficiently close to one, the error norm can become large. However, if our original motivation is the solution of a linear system of equations, say, $Rx = b$, then the error can be improved by resorting to iterative refinement if the matrix R either is given or can be computed accurately from (F, G). In what follows we show that for a diagonal F, the matrix R can be evaluated to high relative accuracy if u_1 and v_1 are exact.

2.12.1 Computing the Matrix R

Given a positive-definite structured matrix R that satisfies

$$R - FRF^T = u_0 u_0^T - v_0 v_0^T$$

with F diagonal and stable, its entries can be computed to high relative accuracy as we explain below.

It follows from the displacement equation that

$$r_{ij} = \frac{u_{i0}u_{j0} - v_{i0}v_{j0}}{1 - f_i f_j} = \frac{u_{i0}u_{j0}\left(1 - \frac{v_{i0}}{u_{i0}}\frac{v_{j0}}{u_{j0}}\right)}{1 - f_i f_j},$$

where, by positive definiteness, the ratios

$$\frac{v_{i0}}{u_{i0}} \text{ and } \frac{v_{j0}}{u_{j0}}$$

are strictly less than one.

The term $\xi = 1 - \frac{v_{i0}}{u_{i0}}\frac{v_{j0}}{u_{j0}}$ can be evaluated to high relative accuracy as explained earlier in Sec. 2.6.4, viz.,

```
If $\frac{v_{i0}}{u_{i0}}\frac{v_{j0}}{u_{j0}} < 1/2$
   then $\xi = 1 - \frac{v_{i0}}{u_{i0}}\frac{v_{j0}}{u_{j0}}$
else
   $d_1 = \frac{|u_{i0}|-|v_{i0}|}{|u_{i0}|}$, $d_2 = \frac{|u_{j0}|-|v_{j0}|}{|u_{j0}|}$,
   $\xi = d_1 + d_2 - d_1 d_2$.
```

Likewise, we evaluate $\mu = (1 - f_i f_j)$ and then $r_{ij} = \frac{u_{i0}u_{j0}\xi}{\mu}$. This guarantees $\hat{r}_{ij} = r_{ij}(1 + O_n(\varepsilon))$.

2.12.2 Iterative Refinement

If the factorization $\hat{L}\hat{L}^T$ is not too inaccurate, and if R is not too ill-conditioned, then it follows from the analysis in [JW77] that the solution $\hat{x}$ of $Rx = b$ can be made backward stable by iterative refinement.

Algorithm 2.12.1 (Iterative Refinement). *Consider a symmetric positive-definite matrix $R \in \mathbb{R}^{n\times n}$ and let $\hat{L}\hat{L}^T$ be a computed Cholesky factorization for it. Let $\hat{x}$ be a computed solution for $Rx = b$. The solution $\hat{x}$ can be made backward stable as follows:*

```
Set $\hat{x}_0 = \hat{x}$, $r = b - R\hat{x}_0$
repeat until $\|r\|_2 \leq O_n(\varepsilon)\|R\|_2\ \|\hat{x}\|_2$
    solve $\hat{L}\hat{L}^T\delta x = r$
    set $\hat{x}_i = \hat{x}_{i-1} + \delta x$
    $r = b - R\hat{x}_i$
end repeat.
```

◇

2.13 ENHANCING THE ROBUSTNESS OF THE ALGORITHM

The performance and robustness of the generalized Schur algorithm can be improved by further enhancements.

Observe that the positive definiteness of R imposes conditions on the columns of the generator matrix:

- For a diagonal and stable F, a necessary condition for the positive definiteness of the matrix is that we must have

$$|\widehat{u}_{i+1}| > |\widehat{v}_{i+1}|, \tag{2.13.1}$$

  where the inequality holds componentwise.

- For a lower triangular contractive F, a necessary condition for positive definiteness is

$$\|u_i\|_2 \geq \|v_i\|_2.$$

- In all cases, the condition $|\Phi_i u_i|_{i+1} > |v_i|_{i+1}$ is required to ensure that the reflection coefficient of the hyperbolic rotation, Θ_i, is less than 1.

We have found that if all these necessary conditions are enforced explicitly, the algorithm is more reliable numerically. An example of this can be found in Sec. 2.13.3.

We now show how the OD and H methods can be modified to preserve the sign of the J-norm of each row of the prearray.

2.13.1 Enhancing the OD Method

The OD method can be enhanced to preserve the sign of the J-norm of the row to which it is applied. More specifically, if the jth row of the perturbed prearray has positive J-norm, then by adding a small perturbation to the jth row of the computed postarray we can guarantee a positive J-norm. If the jth row of the perturbed prearray does not have a positive J-norm, then in general there does not exist a small perturbation for the jth row of the postarray that will guarantee a positive J-norm. For such a row, the prearray must be perturbed to make its J-norm sufficiently positive and then the hyperbolic rotation must be reapplied by the OD method to that row. The new jth row of the postarray can now be made to have a positive J-norm by a small perturbation. The details are given in the algorithm below. For the case of a diagonal and stable F, all the rows of the prearray should have positive J-norm. The algorithm should enforce this property.

In the statement of the algorithm, $\begin{bmatrix} x & y \end{bmatrix}$ stands for a particular row of the prearray $\bar{G}_{i+1}$, $\begin{bmatrix} \widehat{x}_1 & \widehat{y}_1 \end{bmatrix}$ stands for the corresponding row of the postarray $\widehat{G}_{i+1}$, and Θ stands for the hyperbolic rotation. Here we are explicitly assuming that $|x| > |y|$, which is automatically the case when F is diagonal and stable. Otherwise, since $\begin{bmatrix} y & x \end{bmatrix} \Theta = \begin{bmatrix} y_1 & x_1 \end{bmatrix}$, the technique must be used with the elements of the input row interchanged.

Algorithm 2.13.1 (Enhanced OD Method).

```
Assumption: $|x| > |y|$.
if $|\widehat{x}_1| < |\widehat{y}_1|$
  $\gamma_1 = O_n(\varepsilon)(|\widehat{x}_1| + |\widehat{y}_1|)sign(\widehat{x}_1)$
  $\gamma_2 = O_n(\varepsilon)(|\widehat{x}_1| + |\widehat{y}_1|)sign(\widehat{y}_1)$
  if $|\widehat{x}_1 + \gamma_1| > |\widehat{y}_1 - \gamma_2|$ then
    $\widehat{x}_1 = \widehat{x}_1 + \gamma_1$
    $\widehat{y}_1 = \widehat{y}_1 - \gamma_2$
  else
    $\eta_1 = O_n(\varepsilon)(|x| + |y|)sign(x)$
    $\eta_2 = O_n(\varepsilon)(|x| + |y|)sign(y)$
    $\begin{bmatrix} \widehat{x}_1 & \widehat{y}_1 \end{bmatrix} = \begin{bmatrix} x + \eta_1 & y - \eta_2 \end{bmatrix} \Theta$ (via the OD method)
    if $|\widehat{x}_1| > |\widehat{y}_1|$ then $\widehat{x}_1 = \widehat{x}_1$ and $\widehat{y}_1 = \widehat{y}_1$
    else
      $\gamma_1 = O_n(\varepsilon)(|\widehat{x}_1| + |\widehat{y}_1|)sign(\widehat{x}_1)$
      $\gamma_2 = O_n(\varepsilon)(|\widehat{x}_1| + |\widehat{y}_1|)sign(\widehat{y}_1)$
      $\widehat{x}_1 = \widehat{x}_1 + \gamma_1$
      $\widehat{y}_1 = \widehat{y}_1 - \gamma_2$
    endif
  endif
endif
```

◊

2.13.2 Enhancing the H Procedure

Here again, $\begin{bmatrix} x & y \end{bmatrix}$ stands for a particular row of the prearray $\bar{G}_{i+1}$, and $\begin{bmatrix} \widehat{x}_1 & \widehat{y}_1 \end{bmatrix}$ stands for the corresponding row of the postarray. We shall again assume that $|x| > |y|$. If that is not the case then the procedure must be applied to $\begin{bmatrix} y & x \end{bmatrix}$, since $\begin{bmatrix} y & x \end{bmatrix} \Theta = \begin{bmatrix} y_1 & x_1 \end{bmatrix}$.

It follows from $|x| > |y|$ that

$$|\widehat{x}_1 + e_1|^2 - |\widehat{y}_1 + e_2|^2 > 0$$

for the H procedure. Therefore, by adding small numbers to $\widehat{x}_1$ and $\widehat{y}_1$ we can guarantee $|\widehat{x}_1| > |\widehat{y}_1|$.

Algorithm 2.13.2 (Enhanced H Method).

```
Assumption: $|x| > |y|$.
Apply the hyperbolic rotation $\Theta$ to $\begin{bmatrix} x & y \end{bmatrix}$ using the H procedure.
If $|\widehat{x}_1| < |\widehat{y}_1|$ then
  $\widehat{y}_1 = |\widehat{x}_1|(1 - 3\varepsilon)sign(\widehat{y}_1)$.
```

◊

2.13.3 A Numerical Example

The following example exhibits a positive-definite matrix R for which a direct implementation of the generalized Schur algorithm, without the enhancements and modifications proposed herein, breaks down. On the other hand, the modified Schur algorithm enforces positive definiteness and avoids breakdown as the example shows. The data are as follows:

$$G = \begin{bmatrix} 0.29256168393970 & 0 \\ 0.28263551029525 & -0.10728616660709 \\ 0.09633626413940 & 0.01541380240248 \\ 0.06797943459994 & -0.02572176567354 \\ 0.55275012712414 & 0.22069874528633 \\ 0.42631253478657 & 0.06821000412583 \\ 0.50468895704517 & 0.20125628531328 \\ 0.23936358366577 & -0.09527653751206 \\ 0.14608901804405 & 0.02337424345679 \end{bmatrix},$$

$$F = \text{diag} \begin{bmatrix} 0.40000000000000 \\ 0.97781078411630 \\ -0.00000000433051 \\ 0.97646762001746 \\ -0.99577002371173 \\ 0.00000001005313 \\ -0.99285659894698 \\ 0.99789820799463 \\ -0.00000001100000 \end{bmatrix}.$$

A straightforward implementation of the generalized Schur algorithm (i.e., with a naive implementation of the hyperbolic rotation and the Blaschke matrix-vector multiply) breaks down at the eighth step and declares the matrix indefinite.

On the other hand, our implementation, using the enhanced H procedure and the enhanced Blaschke matrix-vector multiply, successfully completes the matrix factorization and yields a relative error

$$\frac{\|R - \hat{L}\hat{L}^T\|_2}{\varepsilon(1 - \|F\|_2^2)^{-2}\|R\|_2} \approx 0.15.$$

Furthermore, the relative backward error $\|R - \hat{L}\hat{L}^T\|_2/\|R\|_2$ is approximately 10^{-11} (using a machine precision of approximately 10^{-16}).

2.14 RESULTS FOR THE POSITIVE-DEFINITE CASE

The general conclusion is the following:

The modified Schur algorithm of this chapter is backward stable for a large class of positive-definite structured matrices. Generally, it is as stable as can be expected from the perturbation analysis of Sec. 2.5.

More specifically,

- If F is strictly lower triangular and contractive (e.g., $F = Z$), then the modified algorithm is backward stable with no generator growth.

- If F is stable, diagonal, and positive, then by reordering the entries of F in increasing order, there will be no generator growth and the algorithm will be as stable as can be expected from Sec. 2.5. In particular, it will be backward stable if $\|F\|_2$ is not too close to one (e.g., $\|F\|_2^2 \leq 1 - \frac{1}{n^2}$).

- In all other cases, we can use the technique outlined in Sec. 2.11 to control the generator growth and make the algorithm as stable as can be expected from Sec. 2.5. In particular, it is backward stable if $\|F\|_2$ is not too close to one (e.g., $\|F\|_2^2 \leq 1 - \frac{1}{n^2}$).
- If R is given or can be computed accurately (e.g., when F is diagonal), iterative refinement can be used to make the algorithm backward stable for the solution of linear equations.

As far as pivoting is concerned, in the diagonal F case, our analysis shows that it is necessary only when $\|F\|_2$ is very close to one.

- If F is positive (or negative), a good strategy is to reorder the entries of F in increasing order of magnitude.
- If F has both positive and negative entries, then our numerical example of Sec. 2.10.1 indicates that pivoting may not help in controlling the growth of the generators.

In our opinion, for positive-definite structured matrices, with diagonal or strictly lower triangular F, the stabilization of the generalized Schur algorithm is critically dependent on the following:

- Proper implementations of the hyperbolic rotations (using the OD or H procedures).
- Proper evaluation of the Blaschke matrix-vector product.
- Enforcing positive definiteness to avoid early breakdowns.
- Controlling the generator growth.

Acknowledgments

The authors wish to thank Professor Thomas Kailath for comments and feedback on an earlier draft of this chapter. They also gratefully acknowledge the support of the National Science Foundation; the work of A. H. Sayed was partially supported by awards MIP-9796147 and CCR-9732376, and the work of S. Chandrasekaran was partially supported by award CCR-9734290.

APPENDIX FOR CHAPTER 2

2.A PSEUDOCODE FOR THE STABLE SCHUR ALGORITHM

We include here a listing of the stabilized Schur algorithm. The program assumes that the input matrix is positive definite and tries to enforce it. The algorithm listed here can be easily modified to test if a structured matrix is positive definite.[6]

The H Procedure

Input data: The ratio *beta/alpha* represents the reflection coefficient, which is smaller than one in magnitude. Also, y/x is assumed to be smaller than one in magnitude.

Output data: The entries $\begin{bmatrix} x_1 & y_1 \end{bmatrix}$ that result by applying a hyperbolic rotation to $\begin{bmatrix} x & y \end{bmatrix}$, with $|x_1| > |y_1|$.

function $\begin{bmatrix} x1, & y1 \end{bmatrix} = h\_procedure(x, y, beta, alpha)$

```
$c = (beta * y)/(alpha * x)$;
if $c < 0.5$
  $xi = 1 - c$;
else
  $d1 = (\text{abs}(alpha) - \text{abs}(beta))/\text{abs}(alpha)$;
  $d2 = (\text{abs}(x) - \text{abs}(y))/\text{abs}(x)$;
  $xi = d1 + d2 - d1 * d2$;
end
$x1 = (\text{abs}(alpha) * x * xi)/\text{sqrt}((alpha - beta) * (alpha + beta))$;
$y1 = x1 - \text{sqrt}((alpha + beta)/(alpha - beta)) * (x - y)$;
if $\text{abs}(x1) < \text{abs}(y1)$
  $y1 = \text{abs}(x1) * (1 - 3 * eps) * \text{sign}(y1)$
end
```

The Blaschke Matrix-Vector Product

We now list the program that computes $\Phi_i u_i$ for both a diagonal F and a strictly lower triangular F.

Remark. In contrast to the description of the generalized Schur algorithm in Alg. 2.4.1, the codes given here work with quantities $\{\Phi_i, u_i, v_i\}$ that are always n-dimensional. The indices also start at 1 rather than 0. Hence, $\{u_i, v_i\}$ are now taken as $n \times 1$ column vectors whose leading $(i-1)$ entries are zero. Also, Φ_i is taken as an $n \times n$ matrix. This convention is used for convenience of description.

- Input: An $n \times n$ stable and diagonal matrix F, a vector u, a vector v (such that $|v| < |u|$), and an index i $(1 \leq i \leq n)$.

- Output: The matrix vector product $z = \Phi_i u$, where $\Phi_i = (I - f_i F)^{-1}(F - f_i I)$, and the vector $ub = (I - f_i F)^{-1} u$.

[6] The codes listed here are in a MATLAB-like language.

```
$function\ \begin{bmatrix} z, & ub \end{bmatrix} = blaschke\_1(F, u, v, i, n)$

    $ub = u;$
    $z = u;$
    for $j = i : n$
       if $F(i,i) * F(j,j) < 0.5$
         $xi = 1/(1 - F(j,j) * F(i,i));$
       else
           $d1 = 1 - \text{abs}(F(i,i));$
           $d2 = 1 - \text{abs}(F(j,j));$
           $xi = 1/(d1 + d2 - d1 * d2);$
       end
       $ub(j) = xi * z(j);$
       $z(j) = (F(j,j) - F(i,i)) * ub(j);$
       if $\text{abs}(z(j)) < \text{abs}(v(j))$
         $z(j) = \text{abs}(v(j)) * (1 + 3 * eps) * \text{sign}(z(j));$
       end
    end
```

For a strictly lower triangular F we use the following.

Input data: An $n \times n$ strictly lower triangular matrix F, a vector u, a vector v (such that $|v| < |u|$), and an index i $(1 \leq i \leq n)$.

Output data: The matrix–vector product $z = Fu$ and $ub = u$.

```
$function\ \begin{bmatrix} z, & ub \end{bmatrix} = blaschke\_2(F, u, v, i, n)$

    $ub = u;$
    $z = F * u;$
    $z(i) = 0;$
    if $\text{abs}(z(i+1)) < \text{abs}(v(i+1))$
      $z(i+1) = \text{abs}(v(i+1)) * (1 + 3 * eps) * \text{sign}(z(i+1));$
    end
```

The Stable Schur Algorithm

We now list two versions of the stable modified Schur algorithm—one for diagonal stable F and the other for strictly lower triangular F.

Input data: An $n \times n$ diagonal and stable matrix F, a generator $G = \begin{bmatrix} u & v \end{bmatrix}$ in proper form (i.e., top entry of v is zero), with column vectors u, v.

Output data: A lower triangular Cholesky factor L such that $\|R - LL^T\|_2$ satisfies (2.9.1).

```
$function\ L = stable\_schur\_1(u, v, F)$

    $n = size(F, 1);$
    for $i = 1 : n - 1$
```

```
        $\begin{bmatrix} u, & ub \end{bmatrix} = blaschke\_1(F, u, v, i, n);$
        $L(:, i) = \mathrm{sqrt}((1 - F(i, i)) * (1 + F(i, i))) * ub;$
        $a = v(i + 1);$
        $b = u(i + 1);$
        for $j = i + 1 : n$
            $\begin{bmatrix} u(j), & v(j) \end{bmatrix} = h\_procedure(u(j), v(j), a, b);$
        end
        $v(i + 1) = 0;$
    end
    $L(n, n) = (1/\mathrm{sqrt}((1 - F(n, n)) * (1 + F(n, n)))) * u(n);$
    $L(1 : n - 1, n) = zeros(n - 1, 1);$
```

- Input: An $n \times n$ strictly lower triangular matrix F, a generator $G = \begin{bmatrix} u & v \end{bmatrix}$ in proper form (i.e., top entry of v is zero), with column vectors u, v.

- Output: A lower triangular Cholesky factor L such that $\|R - LL^T\|_2$ satisfies (2.9.1).

function $L = stable\_schur\_2(u, v, F)$

```
    $n = size(F, 1);$
    for $i = 1 : n - 1$
        $\begin{bmatrix} u, & ub \end{bmatrix} = blaschke\_2(F, u, v, i, n);$
        $L(:, i) = \mathrm{sqrt}((1 - F(i, i)) * (1 + F(i, i))) * ub;$
        $a = v(i + 1);$
        $b = u(i + 1);$
        for $j = i + 1 : n$
            if $\mathrm{abs}(u(j)) > \mathrm{abs}(v(j))$
                $\begin{bmatrix} u(j), & v(j) \end{bmatrix} = h\_procedure(u(j), v(j), a, b);$
            else
                $\begin{bmatrix} temp\_v, & temp\_u \end{bmatrix} = h\_procedure(v(j), u(j), a, b);$
                $v(j) = temp\_v;\ u(j) = temp\_u;$
            endif
        end
        $v(i + 1) = 0;$
    end
    $L(n, n) = (1/\mathrm{sqrt}((1 - F(n, n)) * (1 + F(n, n)))) * u(n);$
    $L(1 : n - 1, n) = zeros(n - 1, 1);$
```