A Superfast Toeplitz Solver with Improved Numerical Stability

Michael Stewart*

May 16, 2003

Abstract

This paper describes a new $O(n \log^3(n))$ solver for the positive definite Toeplitz system $Tx = b$. Instead of computing generators for the inverse of T, the new algorithm adjoins b to T and applies a superfast Schur algorithm to the resulting augmented matrix. The generators of this augmented matrix and its Schur complements are used by a divide-and-conquer block back-substitution routine to complete the solution of the system. The goal is to avoid the well-known numerical instability inherent in explicit inversion. Experiments suggest that the algorithm is backward stable in most cases.

1 Background

We start with the positive definite Toeplitz matrix

$$T = \begin{bmatrix} t_0 & t_1 & \cdots & \cdots & t_{m-1} \\ \bar{t}_1 & t_0 & t_1 & & \vdots \\ \vdots & \bar{t}_1 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & t_1 \\ \bar{t}_{m-1} & \cdots & \cdots & \bar{t}_1 & t_0 \end{bmatrix} \in \mathbb{C}^{m \times m}$$

and the system of equations $Tx = b$. There are several classes of algorithms for solving such systems: slow algorithms requiring $O(n^3)$ unstructured matrix computations, fast $O(n^2)$ algorithms that exploit the Toeplitz structure and superfast algorithms that achieve a complexity strictly less than $O(n^2)$. Examples of fast algorithms include the Schur and Levinson algorithms. Superfast algorithms have been developed in [4, 6, 11, 1, 8, 2].

One way to view the related approaches of [11, 1, 8, 2] is as a divide-and-conquer variant of the $O(n^2)$ Schur algorithm with fast polynomial multiplication via the FFT used to extend computations from submatrices and Schur complements to the full matrix T. The underlying Schur algorithm, along with several generalizations, is numerically stable, [5, 16, 7], but it has not been shown that this stability extends to the superfast Schur algorithm. In fact, the proposed application of the algorithm to linear systems involves computing generators of T^{-1} and then forming $T^{-1}b$ using the FFT. Numerical methods based on explicit inversion are usually unstable, [10]. Experiments presented in §5 show that the superfast Schur algorithm is no exception; it is not a backward stable algorithm.

We will propose an alternative method that parallels the conventional and stable method of triangular factorization and back-substitution. Instead of inverting T we will transform the system $Tx = b$ to

$$\begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} - T_{12}^{\mathrm{H}} T_{11}^{-1} T_{12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 - T_{12}^{\mathrm{H}} T_{11}^{-1} b_1 \end{bmatrix} \qquad (1)$$

*Department of Mathematics and Statistics, Georgia State University, Atlanta Georgia 30303 (mstewart@mathstat.gsu.edu)

and successively solve the two smaller systems

$$(T_{22} - T_{12}^{\mathrm{H}} T_{11}^{-1} T_{12}) x_2 = b_2 - T_{12}^{\mathrm{H}} T_{11}^{-1} b_1, \qquad T_{11} x_1 = b_1 - T_{12} x_2. \tag{2}$$

When dividing the system $Tx = b$ into two systems, it will simplify the presentation to assume that we have divided them in half. Thus T_{11} is $m/2 \times m/2$. We will assume that T so partitioned.

A superfast Schur algorithm applied to the augmented matrix formed from T and b can be used to compute the triangular system (1). The block back-substitution (2) is nothing more than the solution of two smaller Toeplitz-like systems that can be combined to solve the full system using a divide-and-conquer procedure. The multiplication $T_{12}x_2$ can be performed using the FFT.

The resulting algorithm avoids the suspect step of multiplication by T^{-1} but at the cost of increasing the complexity from $O(n \log^2(n))$ to $O(n \log^3(n))$ floating point operations. The increase in complexity occurs because the second system of (2) involves a modified right-hand-side $b_1 - T_{12}x_2$ and consequently a modified augmented matrix. The new augmented matrix requires its own $O(n \log^2(n))$ superfast Schur factorization so that the overall procedure is $O(n \log^3(n))$.

The algorithm of [2] is the model for the derivation of the new algorithm as well as the benchmark for evaluating stability and efficiency. In the remainder of this section we will describe both the algorithm of [2] and the generalized Schur algorithm for a matrix with arbitrary displacement rank. In §2 we show how the same divide-and-conquer idea can be applied to an augmented system that incorporates the right hand side vector b. In §3 we show how the information computed by a superfast block triangularization of the augmented matrix can be used to solve the system $Tx = b$ without the need for explicit matrix inversion. In §4 we evaluate the computational complexity of the algorithm. In §5 we present the result of numerical experiments that demonstrate the improved stability of the algorithm. Finally, in §6 we make some observations on the possibility of a proof of numerical stability and compare the new method to another stabilized superfast algorithm.

1.1 The Generalized Schur Algorithm

A Toeplitz matrix T has an indefinite rank 2 *displacement*

$$T - ZTZ^{\mathrm{H}} = Y \Sigma Y^{\mathrm{H}} \tag{3}$$

where Z is the downshift matrix, $[Z]_{ij} = 1$ if $i - j = 1$ and $[Z]_{ij} = 0$ otherwise, $\Sigma = 1 \oplus -1$ and

$$Y^{\mathrm{H}} = \begin{bmatrix} \sqrt{t_0} & t_1/\sqrt{t_0} & t_2/\sqrt{t_0} & \cdots & t_{n-1}/\sqrt{t_0} \\ 0 & t_1/\sqrt{t_0} & t_2/\sqrt{t_0} & \cdots & t_{n-1}/\sqrt{t_0} \end{bmatrix}.$$

The equation (3) is called a *displacement equation.* The matrix Σ is the *signature* matrix and Y is the *generator* matrix for T. Any matrix for which the displacement has rank significantly lower than n is *Toeplitz-like.*

The generators of a Toeplitz-like matrix are not unique. Given a generator matrix Y and a matrix H satisfying $H\Sigma H^{\mathrm{H}} = \Sigma$ we have

$$(YH)\Sigma(YH)^{\mathrm{H}} = Y(H\Sigma H^{\mathrm{H}})Y^{\mathrm{H}} = Y\Sigma Y^{\mathrm{H}}$$

so that YH is also a generator matrix for T. For general $\Sigma = I_p \oplus -I_q$, matrices H satisfying $H\Sigma H^{\mathrm{H}} = \Sigma$ are known as Σ-unitary. In the particular case $\Sigma = 1 \oplus -1$, all Σ-unitary matrices have the form

$$H = \frac{1}{\sqrt{1 - |\rho|^2}} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} 1 & \overline{\rho} \\ \rho & 1 \end{bmatrix}$$

where $|a| = |b| = 1$, i.e. the Σ-unitary matrices are just the product of hyperbolic rotations and unitary diagonal matrices.

In §2 we will need to consider rank $p+q$ displacements (3) with $\Sigma = I_p \oplus -I_q$. Among the more useful general Σ-unitary transformations are the block diagonal unitary matrices $U \oplus V$ and hyperbolic rotations

$$\begin{bmatrix} I & & & & \\ & \frac{1}{\sqrt{1-|\rho|^2}} & & \frac{\rho}{\sqrt{1-|\rho|^2}} & \\ & & I & & \\ & \frac{\bar{\rho}}{\sqrt{1-|\rho|^2}} & & \frac{1}{\sqrt{1-|\rho|^2}} & \\ & & & & I \end{bmatrix}$$

in which the rotation acts on one index in the positive part of the signature and one in the negative. There is also a hyperbolic version of a Householder transformation, [12, 15].

The product of Σ-unitary matrices can be shown to be Σ-unitary. Thus in applying general Σ-unitary transformations it is natural to decompose them into a product of hyperbolic rotations and block unitary transformations. In this paper we will make use of products of hyperbolic rotations and block diagonal plane rotations. However we will use a somewhat nonstandard signature matrix $\Sigma = 1 \oplus -1 \oplus 1 \oplus -1$ so that the block diagonal unitary rotation matrices become

$$\begin{bmatrix} c_1 & 0 & -\overline{s_1} & 0 \\ 0 & c_2 & 0 & -\overline{s_2} \\ s_1 & 0 & c_1 & 0 \\ 0 & s_2 & 0 & c_2 \end{bmatrix}$$

where c_1 and c_2 are real and nonnegative and $c_1^2 + |s_1|^2 = c_2^2 + |s_2|^2 = 1$. The hyperbolic rotations have the form

$$\begin{bmatrix} \frac{1}{\sqrt{1-|\rho_1|^2}} & 0 & \frac{\rho_1}{\sqrt{1-|\rho_1|^2}} & 0 \\ 0 & \frac{1}{\sqrt{1-|\rho_2|^2}} & 0 & \frac{\rho_2}{\sqrt{1-|\rho_2|^2}} \\ \frac{\overline{\rho_1}}{\sqrt{1-|\rho_1|^2}} & 0 & \frac{1}{\sqrt{1-|\rho_1|^2}} & 0 \\ 0 & \frac{\overline{\rho_2}}{\sqrt{1-|\rho_2|^2}} & 0 & \frac{1}{\sqrt{1-|\rho_2|^2}} \end{bmatrix}.$$

The Schur algorithm is a fast ($O(n^2)$) algorithm for the Cholesky factorization of T. It achieves the reduction in computation by working with the generator matrix instead of on the entire matrix T. Since we will need the generality in §2 we will describe the generalized Schur algorithm for factorization of a displacement rank $p+q$ Toeplitz-like matrix.[1]

We start with a matrix

$$T = \begin{bmatrix} t_0 & t_{21}^{\mathrm{H}} \\ t_{21} & T_{22} \end{bmatrix}$$

satisfying (3) where $\Sigma = I_p \oplus -I_q$. The first step of the generalized Schur algorithm is to transform the matrix Y. We partition Y as

$$Y = \left[\begin{array}{cc|c} y_{11} & y_{12}^{\mathrm{H}} & y_{13}^{\mathrm{H}} \\ y_{21} & Y_{22} & Y_{23} \end{array}\right]$$

[1]In reference to Schur algorithms, the term "generalized" has been used with two distinct meanings. In the sense we are using it here, it refers to a fast algorithm that factors any matrix, not necessarily Toeplitz, satisfying a displacement equation with $\Sigma = I_p \oplus -I_q$. In [2], however, it refers to a superfast algorithm for solving ordinary Toeplitz systems—what we refer to here as the superfast Schur algorithm.

where y_{11} is a scalar and the vertical line marks the boundary between the first p columns and the last q. We then compute a Σ-unitary H as a product of plane rotations and hyperbolic rotations so that

$$\hat{Y} = YH = \left[\begin{array}{cc|c} \hat{y}_{11} & 0 & 0 \\ \hat{y}_{21} & \hat{Y}_{22} & \hat{Y}_{23} \end{array}\right].$$

Thus $\hat{Y}$ is a generator matrix in which only the leading element of the first row is nonzero. Such generators are said to be in *proper form.* The displacement equation (3) implies that

$$\begin{bmatrix} t_0 & t_{21}^{\mathrm{H}} \end{bmatrix} = \hat{y}_{11} \begin{bmatrix} \hat{y}_{11} & \hat{y}_{21}^{\mathrm{H}} \end{bmatrix}$$

so that the first row of $\hat{Y}^{\mathrm{H}}$ is the first row of the Cholesky factor of T. Further if we define

$$T_S = \begin{bmatrix} 0 & 0 \\ 0 & T_{22} - t_{21} t_0^{-1} t_{21}^{\mathrm{H}} \end{bmatrix}, \qquad Y_S = \begin{bmatrix} Z\hat{Y}(:,1) & \hat{Y}(:,2:p+q) \end{bmatrix}$$

then

$$T_S - Z T_S Z^{\mathrm{H}} = Y_S \Sigma Y_S^{\mathrm{H}}.$$

Thus the zero-bordered Schur complement of T inherits the displacement structure of T and its generators are easily determined by the proper form generators for T. The generalized Schur algorithm repeats this process recursively on T_S with generator matrix Y_S to compute successively the rows of the Cholesky factor.

1.2 The Superfast Schur Algorithm

We will now give a description of the superfast Schur algorithm. The presentation here summarizes material from [1, 2]. The main idea behind speeding up the Schur algorithm is to represent the generators as polynomials and then to use fast polynomial multiplication via the FFT to implement the generator transformations of the Schur algorithm. Suppose T is a Toeplitz-like matrix of displacement rank 2 with generators

$$Y = \begin{bmatrix} v_0 & w_0 \\ v_1 & w_1 \\ \vdots & \vdots \\ v_{n-1} & w_{n-1} \end{bmatrix}.$$

We define the polynomial generators

$$Y_0(z) = \begin{bmatrix} v_0(z) & w_0(z) \end{bmatrix}$$

where

$$v_0(z) = v_0 + v_1 z + v_2 z^2 + \cdots + v_{n-1} z^{n-1}$$

and

$$w_0(z) = w_0 + w_1 z + w_2 z^2 + \cdots + w_{n-1} z^{n-1}.$$

Multiplication by z replaces the shift of the first column of Y so that the first step of the Schur algorithm becomes

$$\begin{bmatrix} v_1(z) & w_1(z) \end{bmatrix} = \frac{1}{\sqrt{1-|\rho_1|^2}} \begin{bmatrix} v_0(z) & w_0(z) \end{bmatrix} \begin{bmatrix} 1 & \rho_1 \\ \overline{\rho_1} & 1 \end{bmatrix} \begin{bmatrix} z & 0 \\ 0 & 1 \end{bmatrix}.$$

At step k of the Schur algorithm we have

$$\begin{bmatrix} v_k(z) & w_k(z) \end{bmatrix} = \frac{1}{\sqrt{1-|\rho_k|^2}} \begin{bmatrix} v_{k-1}(z) & w_{k-1}(z) \end{bmatrix} \begin{bmatrix} 1 & \rho_k \\ \overline{\rho_k} & 1 \end{bmatrix} \begin{bmatrix} z & 0 \\ 0 & 1 \end{bmatrix}$$

so that

$$\begin{bmatrix} v_k(z) & w_k(z) \end{bmatrix} = \begin{bmatrix} v_0(z) & w_0(z) \end{bmatrix} \begin{bmatrix} a_k^{(0)}(z) & b_k^{(0)}(z) \\ \tilde{b}_k^{(0)}(z) & \tilde{a}_k^{(0)}(z) \end{bmatrix}$$

where

$$\begin{bmatrix} a_k^{(0)}(z) & b_k^{(0)}(z) \\ \tilde{b}_k^{(0)}(z) & \tilde{a}_k^{(0)}(z) \end{bmatrix} = \left(\prod_{j=1}^{k} \frac{1}{\sqrt{1-|\rho_j|^2}} \right) \begin{bmatrix} z & \rho_1 \\ z\overline{\rho_1} & 1 \end{bmatrix} \begin{bmatrix} z & \rho_2 \\ z\overline{\rho_2} & 1 \end{bmatrix} \cdots \begin{bmatrix} z & \rho_k \\ z\overline{\rho_k} & 1 \end{bmatrix}.$$

It can be shown inductively that

$$\tilde{a}_k^{(0)}(z) = z^k \overline{a}_k^{(0)}(1/z), \qquad \tilde{b}_k^{(0)}(z) = z^k \overline{b}_k^{(0)}(1/z).$$

Hence the product resulting from k steps of the Schur algorithm can be represented by just the *Schur polynomials* $a_k^{(0)}(z)$ and $b_k^{(0)}(z)$.

To represent an arbitrary sequence of k consecutive steps of the Schur algorithm we define

$$\begin{bmatrix} a_k^{(l)}(z) & b_k^{(l)}(z) \\ \tilde{b}_k^{(l)}(z) & \tilde{a}_k^{(l)}(z) \end{bmatrix} = \left(\prod_{j=l+1}^{l+k} \frac{1}{\sqrt{1-|\rho_j|^2}} \right) \begin{bmatrix} z & \rho_{l+1} \\ z\overline{\rho_{l+1}} & 1 \end{bmatrix} \begin{bmatrix} z & \rho_{l+2} \\ z\overline{\rho_2} & 1 \end{bmatrix} \cdots \begin{bmatrix} z & \rho_{l+k} \\ z\overline{\rho_{l+k}} & 1 \end{bmatrix}$$

so that

$$\begin{bmatrix} v_{l+k}(z) & w_{l+k}(z) \end{bmatrix} = \begin{bmatrix} v_l(z) & w_l(z) \end{bmatrix} \begin{bmatrix} a_k^{(l)}(z) & b_k^{(l)}(z) \\ \tilde{b}_k^{(l)}(z) & \tilde{a}_k^{(l)}(z) \end{bmatrix}. \tag{4}$$

Thus $a_k^{(l)}(z)$ and $b_k^{(l)}(z)$ are Schur polynomials that apply k steps of the Schur algorithm, transforming the generator polynomials $v_l(z)$ and $w_l(z)$ into $v_{l+k}(z)$ and $w_{l+k}(z)$. Since they are formed from products of elementary hyperbolic rotations in exactly the same way as $a_k(z)$ and $b_k(z)$ they also satisfy

$$\tilde{a}_k^{(l)}(z) = z^k \overline{a}_k^{(l)}(1/z), \qquad \tilde{b}_k^{(l)}(z) = z^k \overline{b}_k^{(l)}(1/z).$$

Given the Schur polynomials we can perform k steps of the Schur algorithm via the polynomial multiplication (4). If we use the FFT the computational cost of the multiplication will be $O(n\log(n))$. The Schur polynomials can be computed using a divide-and-conquer procedure based on the doubling step

$$\begin{bmatrix} a_{2k}^{(0)}(z) & b_{2k}^{(0)}(z) \\ \tilde{b}_{2k}^{(0)}(z) & \tilde{a}_{2k}^{(0)}(z) \end{bmatrix} = \begin{bmatrix} a_k^{(0)}(z) & b_k^{(0)}(z) \\ \tilde{b}_k^{(0)}(z) & \tilde{a}_k^{(0)}(z) \end{bmatrix} \begin{bmatrix} a_k^{(k)}(z) & b_k^{(k)}(z) \\ \tilde{b}_k^{(k)}(z) & \tilde{a}_k^{(k)}(z) \end{bmatrix}. \tag{5}$$

This equation represents the multiplication of the polynomials for the first k steps of the Schur algorithm with those for the next k to get the polynomials for carrying out $2k$ steps. Again, (5) is just polynomial multiplication which can be carried out with the FFT in $O(n\log(n))$ operations.

Multiplication by the Schur polynomials in (4) increases the degree of the generator polynomials. Since the length of the generator vectors does not increase in the course of applying the Schur algorithm, the higher powers of z are not necessary for computing a factorization. Thus to save memory and computation we should truncate the generator polynomials. For

$$v(z) = v_0 + v_1 z + \cdots + v_{n-1} z^{n-1}$$

we let a superscript (k) for $k < n$ denote the truncation

$$v^{(k)}(z) = v_0 + v_1 z + \cdots + v_{k-1} z^{(k-1)}.$$

Note that this meaning for a superscript applies only to generator polynomials $v(z)$ and $w(z)$, not to the Schur polynomials $a(z)$ and $b(z)$ for which the superscript has a completely different meaning.

Combining (5) with (4) we get a divide-and-conquer algorithm for computing the Schur polynomials $a_k^{(0)}(z)$ and $b_k^{(0)}(z)$.

```
function $[a(z),\ b(z)]$=sfschur($v(z),\ w(z),n$)
        if $n > 1$ then
            $[a^{(0)}_{n/2}(z),\ b^{(0)}_{n/2}(z)] =$ sfschur($v^{(n/2)}(z),\ w^{(n/2)}(z),n/2$)
            $v_{n/2}(z) = v(z)a^{(0)}_{n/2}(z) + w(z)\tilde{b}^{(0)}_{n/2}(z)$
            $w_{n/2}(z) = v(z)b^{(0)}_{n/2}(z) + w(z)\tilde{a}^{(0)}_{n/2}(z)$
            $[a^{(n/2)}_{n/2}(z),\ b^{(n/2)}_{n/2}(z)\ ] =$ sfschur($v^{(n/2)}_{n/2}(z),\ w^{(n/2)}_{n/2}(z),n/2$)
            $a(z) = a^{(0)}_{n/2}(z)a^{(n/2)}_{n/2}(z) + b^{(0)}_{n/2}(z)\tilde{b}^{(n/2)}_{n/2}(z)$
            $b(z) = a^{(0)}_{n/2}(z)b^{(n/2)}_{n/2}(z) + b^{(0)}_{n/2}(z)\tilde{a}^{(n/2)}_{n/2}(z)$
        else
            $\rho = -w(z)/v(z)$
            $a(z) = z/\sqrt{1-|\rho|^2}$
            $b(z) = \rho/\sqrt{1-|\rho|^2}$
        endif
```

The function `sfschur()` takes two generator polynomials of degree $n-1$ representing two generator vectors of length n. The length is passed as a separate parameter. The output polynomials $a(z)$ and $b(z)$ are the Schur polynomials $a_n^{(0)}(z)$ and $b_n^{(0)}(z)$ for applying n steps of the Schur algorithm. The computation is $O(n\log^2(n))$.

Since the the recursive calls to `sfschur()` use the truncated generators $v^{(n/2)}(z)$ and $w^{(n/2)}(z)$, the problem size is halved with each level of depth in the recursion. In the termination case $n = 1$ only one easily computed hyperbolic rotation needs to be applied: if $n = 1$, $v(z)$ and $w(z)$ are constants and the Schur algorithm reduces to the proper form transformation

$$\begin{bmatrix} \tilde{v}(z) & 0 \end{bmatrix} = \frac{1}{\sqrt{1-|\rho|^2}} \begin{bmatrix} v(z) & w(z) \end{bmatrix} \begin{bmatrix} 1 & \rho \\ \overline{\rho} & 1 \end{bmatrix}$$

with $\rho = -w(z)/v(z)$ and

$$\begin{bmatrix} a(z) & b(z) \\ \tilde{b}(z) & \tilde{a}(z) \end{bmatrix} = \frac{1}{\sqrt{1-|\rho|^2}} \begin{bmatrix} z & \rho \\ z\overline{\rho} & 1 \end{bmatrix}.$$

To solve a Toeplitz system we pass the generator polynomials $v(z)$ and $w(z)$ to `sfschur()` to compute the Schur polynomials

$$\begin{bmatrix} a_n^{(0)}(z) & b_n^{(0)}(z) \end{bmatrix} = \texttt{sfschur}(v(z), w(z), n).$$

The inverse matrix T^{-1} is known to be Toeplitz-like. If

$$\phi(z) = \tilde{a}_n^{(0)}(z) + b_n^{(0)}(z), \qquad \tilde{\phi}(z) = a_n^{(0)}(z) + \tilde{b}_n^{(0)}(z)$$

then the pair of polynomials $\phi(z)$ and $\tilde{\phi}(z)$ are polynomial generators for T^{-1}. This fact is expressed by the well-known Gohberg-Semencul formula. Since `sfschur`() gives generators for T^{-1} we can use the FFT to apply T^{-1} to the right-hand-side vector b to solve the system $Tx = b$ using $O(n \log^2(n))$ operations. Since the multiplication by T^{-1} is $O(n \log(n))$, the additional cost of solving the system with a different right-hand-side is $O(n \log(n))$. More details on the use of this algorithm for solving systems can be found in [1, 2].

2 The Augmented System

Instead of factoring just T we will generalize the superfast Schur algorithm to the augmented system

$$M = \begin{bmatrix} T & b \\ b^{\mathrm{H}} & 1 \end{bmatrix}.$$

Suppose T is positive definite of displacement rank 2 and satisfies the displacement equation (3). We extend the displacement equation to

$$M - \begin{bmatrix} Z & 0 \\ 0 & 0 \end{bmatrix} M \begin{bmatrix} Z^{\mathrm{H}} & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} Y\Sigma Y^{\mathrm{H}} & b \\ b^{\mathrm{H}} & 1 \end{bmatrix}.$$

This displacement has a factorization

$$M - \begin{bmatrix} Z & 0 \\ 0 & 0 \end{bmatrix} M \begin{bmatrix} Z^{\mathrm{H}} & 0 \\ 0 & 0 \end{bmatrix} = \hat{Y} \begin{bmatrix} \Sigma & 0 \\ 0 & \Sigma \end{bmatrix} \hat{Y}^{\mathrm{H}}$$

where

$$\hat{Y} = \begin{bmatrix} Y & b & b \\ 0 & 1 & 0 \end{bmatrix}. \tag{6}$$

The generalized Schur algorithm can use any Σ-unitary transformation to put $\hat{Y}$ into proper form. However in generalizing the superfast Schur algorithm it is convenient to use a special transformation that preserves the structure within the generator matrix. In particular, we will use transformations that keep the generator matrices of T and its Schur complements as submatrices of the generator matrices of M and its Schur complements. The resulting algorithm extends but does not otherwise alter the superfast Schur algorithm; it computes every polynomial computed by the superfast Schur algorithm in exactly the same manner in which they are computed by the superfast Schur algorithm. The right-hand-side part of the augmented matrix is handled through the addition of two new Schur polynomials and one new generator polynomial. These additional polynomials depend on the factorization of T, but the computations relating to T, its Schur complements and their generators do not in any way depend on the new polynomials. The use of structured generator transformations reduces the total amount of computation while also making available generators of both T and M.

We assume that at some point in the application of the generalized Schur algorithm to M we have generators of the form

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ v_1 & w_1 & b_1 & b_1 \\ v_2 & w_2 & b_2 & b_2 \\ 0 & 0 & \delta & \delta - 1/\delta \end{bmatrix}$$

where v_1, w_1, b_1 and δ are scalars and v_1 is real and positive. Note that the initial generators (6) are of this form but with no leading zero rows and with $\delta = 1$. Note also that the matrix

$$\begin{bmatrix} v_1 & w_1 \\ v_2 & w_2 \end{bmatrix}$$

is a generator matrix for the Toeplitz-like leading block of M. Thus positive definiteness of T guarantees that $v_1 \neq 0$ and if $\rho = -w_1/v_1$ then $|\rho| < 1$.

We propose a structured transformation to proper form

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ v_1 & w_1 & b_1 & b_1 \\ v_2 & w_2 & b_2 & b_2 \\ 0 & 0 & \delta & \delta - 1/\delta \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-|\rho|^2}} & \frac{\rho}{\sqrt{1-|\rho|^2}} & 0 & 0 \\ \frac{\overline{\rho}}{\sqrt{1-|\rho|^2}} & \frac{1}{\sqrt{1-|\rho|^2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c & 0 & -\overline{s} & 0 \\ 0 & 1 & 0 & 0 \\ s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} \frac{1}{c} & 0 & 0 & \frac{-\overline{s}}{c} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{-s}{c} & 0 & 0 & \frac{1}{c} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \tilde{v}_1 & 0 & 0 & 0 \\ \tilde{v}_2 & \tilde{w}_2 & \tilde{b}_2 & \tilde{b}_2 \\ \tilde{\delta}_1 & 0 & \tilde{\delta} & \tilde{\delta} - 1/\tilde{\delta} \end{bmatrix}.$$

Clearly if $c = \sqrt{1-|s|^2}$ then each of these transformations is Σ-unitary. Multiplying them together gives

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ v_1 & w_1 & b_1 & b_1 \\ v_2 & w_2 & b_2 & b_2 \\ 0 & 0 & \delta & \delta - 1/\delta \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{1-|\rho|^2}} & \frac{\rho}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} \\ \frac{\overline{\rho}}{\sqrt{1-|\rho|^2}} & \frac{1}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} \\ \frac{s}{c} & 0 & c & \frac{-|s|^2}{c} \\ \frac{-s}{c} & 0 & 0 & \frac{1}{c} \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ \tilde{v}_1 & 0 & 0 & 0 \\ \tilde{v}_2 & \tilde{w}_2 & \tilde{b}_2 & \tilde{b}_2 \\ \tilde{\delta}_1 & 0 & \tilde{\delta} & \tilde{\delta} - 1/\tilde{\delta} \end{bmatrix}. \tag{7}$$

We will show that s and ρ can be chosen so that the transformed generators have the form shown in (7). Let

$$\rho = -\frac{w_1}{v_1}$$

where $|\rho| < 1$ and

$$s = \frac{\overline{b_1}}{\sqrt{v_1^2(1-|\rho|^2) + |b_1|^2}}, \qquad c = \sqrt{1-|s|^2}.$$

Since $|\rho| \neq 1$ and $v_1 \neq 0$, $|s| < 1$ so that $0 < c \leq 1$.

The value of ρ has been chosen to zero the w_1 element. In addition to this

$$\tilde{v}_1 = v_1 \frac{1}{\sqrt{1-|\rho|^2}} + w_1 \frac{\overline{\rho}}{\sqrt{1-|\rho|^2}} = v_1\sqrt{1-|\rho|^2} > 0.$$

The cosine c is

$$c = \sqrt{1-|s|^2} = \sqrt{\frac{v_1^2(1-|\rho|^2)}{v_1^2(1-|\rho|^2) + |b_1|^2}} = \frac{\tilde{v}_1}{\sqrt{v_1^2(1-|\rho|^2) + |b_1|^2}}.$$

The first b_1 transforms to

$$cb_1 - \overline{s}\left(v_1 \frac{1}{\sqrt{1-|\rho|^2}} + w_1 \frac{\overline{\rho}}{\sqrt{1-|\rho|^2}}\right) = cb_1 - \overline{s}\tilde{v}_1 = 0$$

and the second transforms to

$$\frac{1}{c}b_1 - \frac{|s|^2}{c}b_1 - \overline{s}\left(v_1\frac{1}{\sqrt{1-|\rho|^2}} + w_1\frac{\overline{\rho}}{\sqrt{1-|\rho|^2}}\right) = cb_1 - \overline{s}\tilde{v}_1 = 0.$$

Similarly the first and second b_2 element are transformed to the same vector

$$\tilde{b}_2 = cb_2 - \overline{s}\left(v_2\frac{1}{\sqrt{1-|\rho|^2}} + w_2\frac{\overline{\rho}}{\sqrt{1-|\rho|^2}}\right).$$

The presence of the zero element on the bottom row is obvious. The element corresponding to δ is

$$\tilde{\delta} = c\delta. \tag{8}$$

Finally the element corrsponding to $\delta - 1/\delta$ is

$$-\frac{|s|^2}{c}\delta + (\delta - 1/\delta)\frac{1}{c} = \frac{1-c^2}{c}\delta + -\frac{1}{\delta c} = \delta c - \frac{1}{\delta c} = \tilde{\delta} - \frac{1}{\tilde{\delta}}.$$

This verifies that all elements of the transformed generator matrix are as shown. In particular we have shown that if the generators are

$$\begin{bmatrix} v & w & b & b \\ 0 & 0 & \delta & \delta - 1/\delta \end{bmatrix}. \tag{9}$$

then (7) puts the transformed generators in the form

$$\begin{bmatrix} \tilde{v} & \tilde{w} & \tilde{b} & \tilde{b} \\ \tilde{\delta}_1 & 0 & \tilde{\delta} & \tilde{\delta} - 1/\tilde{\delta} \end{bmatrix}. \tag{10}$$

To show that at every stage of the Schur algorithm the generators have the form (9) we first note that the initial generators (6) are of this form. As shown in (7) the generator transformations preserve the structure except for the addition of a nonzero δ_1. However in computing the generators of a Schur complement the first column of the transformed generator matrix (10) is multiplied by

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

which zeros the element δ_1.

In addition to the preservation of the pattern of repeated vectors in the generators, (7) implies that for the first two columns of the transformed generator matrix

$$\begin{bmatrix} 0 & 0 \\ \tilde{v}_1 & 0 \\ \tilde{v}_2 & \tilde{w}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ v_1 & w_1 \\ v_2 & w_2 \end{bmatrix} \begin{bmatrix} 1 & \rho \\ \overline{\rho} & 1 \end{bmatrix} / \sqrt{1-|\rho|^2}.$$

These two columns are no different than if we had simply applied the 2×2 hyperbolic rotation from the ordinary Schur algorithm to the first two columns the generator matrix. The result is that, in applying this form of the generalized Schur algorithm, the first two columns of the generator matrix will be generators of T and its Schur complements.

As in the superfast Schur algorithm, the generators and generator transformations can be represented by polynomials. We represent all but the last row of the generator matrix by three polynomials

$$\begin{bmatrix} v_k(z) & w_k(z) & \beta_k(z) & \beta_k(z) \end{bmatrix} = \begin{bmatrix} 1 & z & \cdots & z^{n-1} \end{bmatrix} \begin{bmatrix} v & w & b & b \end{bmatrix}$$

so that

$$\begin{bmatrix} v_k(z) & w_k(z) & \beta_k(z) & \beta_k(z) \end{bmatrix} = \begin{bmatrix} v_{k-1}(z) & w_{k-1}(z) & \beta_{k-1}(z) & \beta_{k-1}(z) \end{bmatrix} \cdot$$
$$\begin{bmatrix} z\frac{1}{\sqrt{1-|\rho|^2}} & \frac{\rho}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} \\ z\frac{\overline{\rho}}{\sqrt{1-|\rho|^2}} & \frac{1}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} \\ z\frac{s}{c} & 0 & c & \frac{-|s|^2}{c} \\ z\frac{-s}{c} & 0 & 0 & \frac{1}{c} \end{bmatrix}.$$

The scalar δ will not be incorporated into any polynomial; it will be stored and kept track of separately from $v(z)$, $w(z)$ and $\beta(z)$.

As in the displacement rank 2 case, accumulating the product of such transformations results in a special structure. We will show that k steps of the polynomial version of the generalized Schur algorithm have the form

$$\begin{bmatrix} v_{l+k}(z) & w_{l+k}(z) & \beta_{l+k}(z) & \beta_{l+k}(z) \end{bmatrix} = \begin{bmatrix} v_l(z) & w_l(z) & \beta_l(z) & \beta_l(z) \end{bmatrix} \cdot$$
$$\begin{bmatrix} a_k^{(l)}(z) & b_k^{(l)}(z) & c_k^{(l)}(z) & c_k^{(l)}(z) \\ \tilde{b}_k^{(l)}(z) & \tilde{a}_k^{(l)}(z) & d_k^{(l)}(z) & d_k^{(l)}(z) \\ e_k^{(l)}(z) & f_k^{(l)}(z) & g_k^{(l)}(z) & g_k^{(l)}(z) - 1/g_k^{(l)}(0) \\ -e_k^{(l)}(z) & -f_k^{(l)}(z) & -g_k^{(l)}(z) + g_k^{(l)}(0) & -g_k^{(l)}(z) + 1/g_k^{(l)}(0) + g_k^{(l)}(0) \end{bmatrix} \tag{11}$$

This is verified inductively in the following theorem.

Theorem 1 *The product of k matrices of the form*

$$\begin{bmatrix} z\frac{1}{\sqrt{1-|\rho|^2}} & \frac{\rho}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} \\ z\frac{\overline{\rho}}{\sqrt{1-|\rho|^2}} & \frac{1}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} \\ z\frac{s}{c} & 0 & c & \frac{-|s|^2}{c} \\ z\frac{-s}{c} & 0 & 0 & \frac{1}{c} \end{bmatrix} \tag{12}$$

has the form

$$\begin{bmatrix} a_k(z) & b_k(z) & c_k(z) & c_k(z) \\ \tilde{b}_k(z) & \tilde{a}_k(z) & d_k(z) & d_k(z) \\ e_k(z) & f_k(z) & g_k(z) & g_k(z) - 1/g_k(0) \\ -e_k(z) & -f_k(z) & -g_k(z) + g_k(0) & -g_k(z) + 1/g_k(0) + g_k(0) \end{bmatrix}$$

where $e(0) = f(0) = 0$ and where $\tilde{a}(z) = z^k\overline{a}(1/z)$ and $\tilde{b}(z) = z^k\overline{b}(1/z)$.

Proof: We note that the single transformation (12) has the specified form. We assume that the theorem holds for a product of $k-1$ transformations and write

$$\begin{bmatrix} a_k(z) & b_k(z) & c_k(z) & c_k(z) \\ \tilde{b}_k(z) & \tilde{a}_k(z) & d_k(z) & d_k(z) \\ e_k(z) & f_k(z) & g_k(z) & g_k(z) - 1/g_k(0) \\ -e_k(z) & -f_k(z) & -g_k(z) + g_k(0) & -g_k(z) + 1/g_k(0) + g_k(0) \end{bmatrix} =$$

$$\begin{bmatrix} a_{k-1}(z) & b_{k-1}(z) & c_{k-1}(z) & c_{k-1}(z) \\ \tilde{b}_{k-1}(z) & \tilde{a}_{k-1}(z) & d_{k-1}(z) & d_{k-1}(z) \\ e_{k-1}(z) & f_{k-1}(z) & g_{k-1}(z) & g_{k-1}(z) - 1/g_{k-1}(0) \\ -e_{k-1}(z) & -f_{k-1}(z) & -g_{k-1}(z) + g_{k-1}(0) & -g_{k-1}(z) + 1/g_{k-1}(0) + g_{k-1}(0) \end{bmatrix} \cdot$$

$$\begin{bmatrix} z\frac{1}{\sqrt{1-|\rho|^2}} & \frac{\rho}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s}}{\sqrt{1-|\rho|^2}} \\ z\frac{\overline{\rho}}{\sqrt{1-|\rho|^2}} & \frac{1}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} & \frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}} \\ z\frac{s}{c} & 0 & c & \frac{-|s|^2}{c} \\ z\frac{-s}{c} & 0 & 0 & \frac{1}{c} \end{bmatrix}.$$

This gives two relations for each of the polynomials $e_k(z)$, $f_k(z)$, $c_k(z)$ and $d_k(z)$. The relations for $e_k(z)$ and $-e_k(z)$ (obtained by computing the $(3,1)$ and $(4,1)$ elements of the left-hand-side) are

$$e_k(z) = \frac{z}{\sqrt{1-|\rho|^2}} e_{k-1}(z) + \frac{z\overline{\rho}}{\sqrt{1-|\rho|^2}} f_{k-1}(z) + \frac{zs}{c} g_{k-1}(z) - \frac{zs}{c} g_{k-1}(z) + \frac{zs}{c}\frac{1}{g_{k-1}(0)}$$

and

$$-e_k(z) = -\frac{z}{\sqrt{1-|\rho|^2}} e_{k-1}(z) - \frac{z\overline{\rho}}{\sqrt{1-|\rho|^2}} f_{k-1}(z) - \frac{zs}{c} g_{k-1}(z) + \frac{zs}{c} g_{k-1}(0) + \frac{zs}{c} g_{k-1}(z) - \frac{zs}{c}\frac{1}{g_{k-1}(0)} - \frac{zs}{c} g_{k-1}(0).$$

These two relations clearly define the same polynomial $e_k(z)$. Verification that the two relations for $f_k(z)$ give the same polynomial is similar, as is the verification for $c_k(z)$ and $d_k(z)$.

Every term in the expression for $e_k(z)$ has z as a factor. Thus $e_k(0) = 0$. Since

$$f_k(z) = e_{k-1}(z)\frac{\rho}{\sqrt{1-|\rho|^2}} + f_{k-1}(z)\frac{1}{\sqrt{1-|\rho|^2}}$$

we see that $f_k(0) = 0$ follows from $e_{k-1}(0) = 0$ and $f_{k-1}(0) = 0$.

The relations $\tilde{a}_k(z) = z^k\overline{a}_k(1/z)$ and $\tilde{b}_k(z) = z^k\overline{a}_k(1/z)$ follow from

$$\begin{bmatrix} a_k(z) & b_k(z) \\ \tilde{b}_k(z) & \tilde{a}_k(z) \end{bmatrix} = \begin{bmatrix} a_{k-1}(z) & b_{k-1}(z) \\ \tilde{b}_{k-1}(z) & \tilde{a}_{k-1}(z) \end{bmatrix} \begin{bmatrix} z & \rho \\ z\overline{\rho} & 1 \end{bmatrix} / \sqrt{1-|\rho|^2}$$

in exactly the same way as this result follows for the displacement rank 2 case.

To verify the identities for the polynomial $g_k(z)$ we note that if we define

$$h_k(z) = e_{k-1}(z)\frac{-\overline{s}}{\sqrt{1-|\rho|^2}} + f_{k-1}(z)\frac{-\overline{s\rho}}{\sqrt{1-|\rho|^2}}$$

then $h_k(0) = 0$ since $e_{k-1}(0) = f_{k-1}(0) = 0$. In terms of $h_k(z)$ the lower right 2×2 block is

$$\begin{bmatrix} g_k(z) & g_k(z) - 1/g_k(0) \\ -g_k(z) + g_k(0) & -g_k(z) + 1/g_k(0) + g_k(0) \end{bmatrix} =$$

$$\begin{bmatrix} h_k(z) + cg_{k-1}(z) & h_k(z) + cg_{k-1}(z) - 1/(cg_{k-1}(0)) \\ -h_k(z) - cg_{k-1}(z) + cg_{k-1}(0) & -h_k(z) - cg_{k-1}(z) + cg_{k-1}(0) + 1/(cg_{k-1}(0)) \end{bmatrix}$$

where

$$g_k(z) = h_k(z) + cg_{k-1}(z)$$

and

$$g_k(0) = cg_{k-1}(0) \tag{13}$$

since $h_k(0) = 0$. ■

Having established the form of the generator transformations we construct a superfast version of the Schur algorithm for the augmented matrix using a doubling relation analogous to (5)

$$\begin{bmatrix} a_{2k}^{(0)}(z) & b_{2k}^{(0)}(z) & c_{2k}^{(0)}(z) & c_{2k}^{(0)}(z) \\ \tilde{b}_{2k}^{(0)}(z) & \tilde{a}_{2k}^{(0)}(z) & d_{2k}^{(0)}(z) & d_{2k}^{(0)}(z) \\ e_{2k}^{(0)}(z) & f_{2k}^{(0)}(z) & g_{2k}^{(0)}(z) & g_{2k}^{(0)}(z) - 1/g_{2k}^{(0)}(0) \\ -e_{2k}^{(0)}(z) & -f_{2k}^{(0)}(z) & -g_{2k}^{(0)}(z) + g_{2k}^{(0)}(0) & -g_{2k}^{(0)}(z) + 1/g_{2k}^{(0)}(0) + g_{2k}^{(0)}(0) \end{bmatrix} =$$
$$\begin{bmatrix} a_{k}^{(0)}(z) & b_{k}^{(0)}(z) & c_{k}^{(0)}(z) & c_{k}^{(0)}(z) \\ \tilde{b}_{k}^{(0)}(z) & \tilde{a}_{k}^{(0)}(z) & d_{k}^{(0)}(z) & d_{k}^{(0)}(z) \\ e_{k}^{(0)}(z) & f_{k}^{(0)}(z) & g_{k}^{(0)}(z) & g_{k}^{(0)}(z) - 1/g_{k}^{(0)}(0) \\ -e_{k}^{(0)}(z) & -f_{k}^{(0)}(z) & -g_{k}^{(0)}(z) + g_{k}^{(0)}(0) & -g_{k}^{(0)}(z) + 1/g_{k}^{(0)}(0) + g_{k}^{(0)}(0) \end{bmatrix} \cdot$$
$$\begin{bmatrix} a_{k}^{(k)}(z) & b_{k}^{(k)}(z) & c_{k}^{(k)}(z) & c_{k}^{(k)}(z) \\ \tilde{b}_{k}^{(k)}(z) & \tilde{a}_{k}^{(k)}(z) & d_{k}^{(k)}(z) & d_{k}^{(k)}(z) \\ e_{k}^{(k)}(z) & f_{k}^{(k)}(z) & g_{k}^{(k)}(z) & g_{k}^{(k)}(z) - 1/g_{k}^{(k)}(0) \\ -e_{k}^{(k)}(z) & -f_{k}^{(k)}(z) & -g_{k}^{(k)}(z) + g_{k}^{(k)}(0) & -g_{k}^{(k)}(z) + 1/g_{k}^{(k)}(0) + g_{k}^{(k)}(0) \end{bmatrix} . \tag{14}$$

It turns out that a complete algorithm for the solution of $Tx = b$ can be formulated without computing $e_k(z)$, $f_k(z)$ and $g_k(z)$. We will however need $a_k(z)$, $b_k(z)$, $c_k(z)$, $d_k(z)$ and the scalar $g_k(0)$. The updates from (14) that we will use include the Schur polynomial computation

$$\begin{bmatrix} a_{2k}^{(0)}(z) & b_{2k}^{(0)}(z) \\ \tilde{b}_{2k}^{(0)}(z) & \tilde{a}_{2k}^{(0)}(z) \end{bmatrix} = \begin{bmatrix} a_{k}^{(0)}(z) & b_{k}^{(0)}(z) \\ \tilde{b}_{k}^{(0)}(z) & \tilde{a}_{k}^{(0)}(z) \end{bmatrix} \begin{bmatrix} a_{k}^{(k)}(z) & b_{k}^{(k)}(z) \\ \tilde{b}_{k}^{(k)}(z) & \tilde{a}_{k}^{(k)}(z) \end{bmatrix}, \tag{15}$$

and the updates for $c(z)$ and $d(z)$

$$c_{2k}^{(0)}(z) = a_k^{(0)}(z)c_k^{(k)}(z) + b_k^{(0)}(z)d_k^{(k)}(z) + g_k^{(k)}(0)c_k^{(0)}(z) \tag{16}$$

$$d_{2k}^{(0)}(z) = \tilde{b}_k^{(0)}(z)c_k^{(k)}(z) + \tilde{a}_k^{(0)}(z)d_k^{(k)}(z) + g_k^{(k)}(0)d_k^{(0)}(z). \tag{17}$$

Since $e(0) = f(0) = 0$

$$g_{2k}^{(0)}(z) = e_k^{(0)}(z)c_k^{(k)}(z) + f_k^{(0)}(z)d_k^{(k)}(z) + g_k^{(0)}(z)g_k^{(k)}(z) + \\ (g_k^{(0)}(z) - 1/g_k^{(0)}(0))(-g_k^{(k)}(z) + g_k^{(k)}(0))$$

gives the scalar update

$$g_{2k}^{(0)}(0) = g_k^{(0)}(0)g_k^{(k)}(0). \tag{18}$$

From (11) we use the generator transformations

$$v_{k/2}(z) = v_0(z)a_{k/2}^{(0)}(z) + w_0(z)\tilde{b}_{k/2}^{(0)}(z)$$

and

$$w_{k/2}(z) = v_0(z)b^{(0)}_{k/2}(z) + w_0(z)\tilde{a}^{(0)}_{k/2}(z).$$

And for $\beta(z)$ we use

$$\beta_{k/2}(z) = v_0(z)c^{(0)}_{k/2}(z) + w_0(z)d^{(0)}_{k/2}(z) + \beta_0(z)g^{(0)}_{k/2}(0).$$

We keep track of the quantity δ in the augmented matrix generators by noting that if $\delta_0 = 1$ and δ_k represents the quantity δ after k steps of the Schur algorithm then by comparing (8) and (13) we see that both δ_k and $g^{(0)}_k(0)$ are products of the cosines c used in the generalized Schur factorization of the augmented matrix. It follows that

$$\delta_k = g^{(0)}_k(0).$$

In the algorithm we will use the relation

$$\delta_{l+k/2} = g^{(l)}_{k/2}(0)\delta_l.$$

Since we will need only $g^{(l)}_k(0)$ and not the full polynomial $g^{(l)}_k(z)$ we set $g^{(l)}_k = g^{(l)}_k(0)$. Putting everything together we get the superfast generalized Schur algorithm for the augmented matrix.

```
function $[a(z),\ b(z),\ c(z),\ d(z),\ g,\ S(z),\ P(z)]=$ bsfschur($v(z),\ w(z),\beta(z),\ \delta,\ n$)
        if $n > 1$ then
            $[a^{(0)}_{n/2}(z),\ b^{(0)}_{n/2}(z),\ c^{(0)}_{n/2}(z),\ d^{(0)}_{n/2}(z),\ g^{(0)}_{n/2},\ S^{(0)}_{n/2}(z),\ P^{(n/2)}_{n/2}\ ] =$
                        bsfschur($v^{(n/2)}(z),\ w^{(n/2)}(z),\ \beta^{(n/2)}(z),\ \delta,\ n/2$)
            $v_{n/2}(z) = v(z)a^{(0)}_{n/2}(z) + w(z)\tilde{b}^{(0)}_{n/2}(z)$
            $w_{n/2}(z) = v(z)b^{(0)}_{n/2}(z) + w(z)\tilde{a}^{(0)}_{n/2}(z)$
            $\beta_{n/2}(z) = v(z)c^{(0)}_{n/2}(z) + w(z)d^{(0)}_{n/2}(z) + \beta(z)g^{(0)}_{n/2}$
            $[a^{(n/2)}_{n/2}(z),\ b^{(n/2)}_{n/2}(z),\ c^{(n/2)}_{n/2}(z),\ d^{(n/2)}_{n/2}(z),\ g^{(n/2)}_{n/2},\ S^{(n/2)}_{n/2}(z),\ P^{(n/2)}_{n/2}(z)\ ] =$
                        bsfschur($v^{(n/2)}_{n/2}(z),\ w^{(n/2)}_{n/2}(z),\ \beta^{(n/2)}_{n/2}(z),\ \delta g^{(0)}_{n/2},\ n/2$)
            $a(z) = a^{(0)}_{n/2}(z)a^{(n/2)}_{n/2}(z) + b^{(0)}_{n/2}(z)\tilde{b}^{(n/2)}_{n/2}(z)$
            $b(z) = a^{(0)}_{n/2}(z)b^{(n/2)}_{n/2}(z) + b^{(0)}_{n/2}(z)\tilde{a}^{(n/2)}_{n/2}(z)$
            $c(z) = a^{(0)}_{n/2}(z)c^{(n/2)}_{n/2}(z) + b^{(0)}_{n/2}(z)d^{(n/2)}_{n/2}(z) + g^{(n/2)}_{n/2}c^{(0)}_{n/2}(z)$
            $d(z) = \tilde{b}^{(0)}_{n/2}(z)c^{(n/2)}_{n/2}(z) + \tilde{a}^{(0)}_{n/2}(z)d^{(n/2)}_{n/2}(z) + g^{(n/2)}_{n/2}d^{(0)}_{n/2}(z)$
            $g = g^{(0)}_{n/2}g^{(n/2)}_{n/2}$
            $S(z) = \begin{bmatrix} v(z) & w(z) & \beta(z)/\delta \end{bmatrix}$
            $S(z) = \begin{bmatrix} S(z) \\ S^{(0)}_{n/2}(z) \\ S^{(n/2)}_{n/2}(z) \end{bmatrix}$
            $P(z) = \begin{bmatrix} a(z) & b(z) \end{bmatrix}$
            $P(z) = \begin{bmatrix} P(z) \\ P^{(0)}_{n/2}(z) \\ P^{(n/2)}_{n/2}(z) \end{bmatrix}$
        else
            $\rho = -w(z)/v(z)$
            $a(z) = z/\sqrt{1-|\rho|^2}$
        $b(z) = \rho/\sqrt{1-|\rho|^2}$
        $s = \overline{\beta(z)}/\sqrt{(1-|\rho|^2)|v(z)|^2 + |\beta(z)|^2}$
        $c(z) = -\overline{s}/\sqrt{1-|\rho|^2}$
        $d(z) = -\overline{s\rho}/\sqrt{1-|\rho|^2}$
        $g = \sqrt{1-|s|^2}$
        $S(z) = \begin{bmatrix} v(z) & w(z) & \beta(z)/\delta \end{bmatrix}$
        $P(z) = \begin{bmatrix} a(z) & b(z) \end{bmatrix}$
    endif
```

Several features of the algorithm need to be explained. The function `bsfschur`() takes generator polynomials $v(z)$, $w(z)$ and $\beta(z)$, the scalar parameter δ and the integer parameter n where $n-1$ is the degree of the polynomials $v(z)$, $w(z)$ and $\beta(z)$. The parameter n is also the number of Schur steps to be performed. The algorithm is recursive and uses the doubling recurrence for the polynomials $a(z)$, $b(z)$, $c(z)$ and $d(z)$. The recursion terminates when $n = 1$. This corresponds to a 1×1 Toeplitz-like matrix or a 2×2 augmented matrix. When $n = 1$ the polynomials are just the elements of the matrix in (7).

The function `bsfschur`() incorporates code to compute and store generators for the Schur complements of the augmented matrix and its Schur complements in $S(z)$ and the corresponding Schur polynomials in $P(z)$. To understand the storage scheme, partition an $(n+1) \times (n+1)$ augmented matrix as

$$M = \begin{bmatrix} T & b \\ b^{\mathrm{H}} & 2 - \frac{1}{\delta^2} \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} & b_1 \\ T_{12}^{\mathrm{H}} & T_{22} & b_2 \\ b_1^{\mathrm{H}} & b_2 & 2 - \frac{1}{\delta^2} \end{bmatrix}.$$

Suppose M has generators

$$\begin{bmatrix} v & w & b\delta & b\delta \\ 0 & 0 & \delta & \delta - 1/\delta \end{bmatrix}$$

with

$$\begin{bmatrix} v(z) & w(z) & \beta(z) \end{bmatrix} = \begin{bmatrix} 1 & z & \cdots & z^{n-1} \end{bmatrix} \begin{bmatrix} v & w & b\delta \end{bmatrix}.$$

Thus $v(z)$ and $w(z)$ are generators of T and the coefficients of $\beta(z)/\delta$ are the elements of the vector b. If `sfschur`() is run on generators for M then the matrix $S(z)$ is constructed recursively as

$$S(z) = S_0^{(0)}(z) = \begin{bmatrix} v(z) \quad w(z) \quad \beta(z)/\delta \\ S_{n/2}^{(0)}(z) \\ S_{n/2}^{(n/2)}(z) \end{bmatrix}. \tag{19}$$

The matrices $S_{n/2}^{(0)}(z)$ and $S_{n/2}^{(n/2)}$ are defined in the same way but for the submatrix

$$M_1 = \begin{bmatrix} T_{11} & b_1 \\ b_1^{\mathrm{H}} & 2 - \frac{1}{\delta^2} \end{bmatrix}.$$

and for the Schur complement

$$M_S = \begin{bmatrix} T_{22} - T_{12}^{\mathrm{H}} T_{11}^{-1} T_{12} & b_2 - T_{12}^{\mathrm{H}} T_{11}^{-1} b_1 \\ b_2^{\mathrm{H}} - b_1^{\mathrm{H}} T_{11}^{-1} T_{12} & 2 - \frac{1}{\delta^2} - b_1^{\mathrm{H}} T_{11}^{-1} b_1 \end{bmatrix}.$$

This recursive definition of $S(z)$ terminates when the relevant augmented matrix is 2×2 in which case

$$S(z) = \begin{bmatrix} v(z) & w(z) & \beta(z)/\delta \end{bmatrix}.$$

The structure of $P(z)$ is defined in a similar manner

$$P(z) = P_0^{(0)}(z) = \begin{bmatrix} a_0^{(0)}(z) \quad b_0^{(0)}(z) \\ P_{n/2}^{(0)}(z) \\ P_{n/2}^{(n/2)}(z) \end{bmatrix}. \tag{20}$$

The information contained in $S(z)$ and $P(z)$ will be used by a function that solves the system $Tx = b$ or by a function to recompute the polynomials associated with a different right-hand-side.

Given the Schur polynomials in $P(z)$ and the generators in $S(z)$, it is possible to recompute $c(z)$, $d(z)$, g and $\beta(z)$ for a different right hand side without repeating the computation of $v(z)$, $w(z)$, $a(z)$ and $b(z)$. In the following we assume that the inputs $S_0^{(0)}(z)$ and $P_0^{(0)}(z)$ are partitioned as in (19) and (20). The elements of the new right-hand-side vector are the coefficients of the input polynomial $\beta(z)/\delta$. The outputs are the new polynomials $c_n^{(0)}(z)$ and $d_n^{(0)}(z)$, the scalar $g_n^{(0)}$ and $S_n^{(0)}(z)$ updated with the new right-hand-side.

```
function $[c(z),\ d(z),\ g,\ S(z)] = \texttt{rhs}(S_0^{(0)}(z),\ P_0^{(0)}(z),\ \beta(z),\ \delta,\ n)$
        if $n > 1$ then
            $[c_{n/2}^{(0)}(z)\ ,\ d_{n/2}^{(0)}(z),\ g_{n/2}^{(0)},\ S_{n/2}^{(0)}(z)\ ] =$
                        $\texttt{rhs}(S_{n/2}^{(0)},\ P_{n/2}^{(0)},\ \beta^{(n/2)}(z),\ \delta,\ n/2)$
            $\beta_{n/2}(z) = v_0(z)c_{n/2}^{(0)}(z) + w_0(z)d_{n/2}^{(0)}(z) + \beta(z)g_{n/2}^{(0)}$
            $[c_{n/2}^{(n/2)}(z),\ d_{n/2}^{(n/2)}(z),\ g_{n/2}^{(n/2)},\ S_{n/2}^{(n/2)}(z)\ ] =$
                        $\texttt{rhs}(S_{n/2}^{(n/2)},\ P_{n/2}^{(n/2)},\ \beta_{n/2}^{(n/2)}(z),\ \delta g_{n/2}^{(0)},\ n/2)$
            $c(z) = a_{n/2}^{(0)}(z)c_{n/2}^{(n/2)}(z) + b_{n/2}^{(0)}(z)d_{n/2}^{(n/2)}(z) + g_{n/2}^{(n/2)}c_{n/2}^{(0)}(z)$
            $d(z) = \tilde{b}_{n/2}^{(0)}(z)c_{n/2}^{(n/2)}(z) + \tilde{a}_{n/2}^{(0)}(z)d_{n/2}^{(n/2)}(z) + g_{n/2}^{(n/2)}d_{n/2}^{(0)}(z)$
            $g = g_{n/2}^{(0)}g_{n/2}^{(n/2)}$
            $S(z) = \begin{bmatrix} v(z) & w(z) & \beta(z)/\delta \end{bmatrix}$
            $S(z) = \begin{bmatrix} S(z) \\ S_{n/2}^{(0)}(z) \\ S_{n/2}^{(n/2)}(z) \end{bmatrix}$
        else
            $\rho = -w_0(z)/v_0(z)$
            $s = \overline{\beta(z)}/\sqrt{(1-|\rho|^2)|v(z)|^2 + |\beta(z)|^2}$
            $c(z) = -\overline{s}/\sqrt{1-|\rho|^2}$
            $d(z) = -\overline{s\rho}/\sqrt{1-|\rho|^2}$
            $g = \sqrt{1-|s|^2}$
            $S(z) = \begin{bmatrix} v(z) & w(z) & \beta(z)/\delta \end{bmatrix}$
        endif
```

Note that there is no $P(z)$ as output for `rhs()`. This is because $P(z)$ does not depend on the right-hand-side. The use of function `rhs()` substantially reduces the computation for problems that involve multiple right-hand-sides. More significantly, as we will see in the next section, it allows us to efficiently deal with transformations of the right-hand-side when solving the system $Tx = b$.

3 Divide-and-Conquer Back-Substitution

The function `bsfschur`() is a divide-and-conquer procedure for factoring the $(n+1) \times (n+1)$ augmented matrix

$$M = \begin{bmatrix} T & b \\ b^{\mathrm{H}} & 1 \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} & b_1 \\ T_{12}^{\mathrm{H}} & T_{22} & b_2 \\ b_1^{\mathrm{H}} & b_2^{\mathrm{H}} & 1 \end{bmatrix}. \tag{21}$$

After $n/2$ steps of elimination on this matrix we have the factorization

$$M = \begin{bmatrix} I & 0 & 0 \\ T_{12}^{\mathrm{H}} T_{11}^{-1} & I & 0 \\ b_1^{\mathrm{H}} T_{11}^{-1} & 0 & 1 \end{bmatrix} \begin{bmatrix} T_{11} & 0 & 0 \\ 0 & T_{22} - T_{12}^{\mathrm{H}} T_{11}^{-1} T_{12} & b_2 - T_{12}^{\mathrm{H}} T_{11}^{-1} b_1 \\ 0 & b_2^{\mathrm{H}} - b_1^{\mathrm{H}} T_{11}^{-1} T_{12} & 1 - b_1^{\mathrm{H}} T_{11}^{-1} b_1 \end{bmatrix} \cdot \begin{bmatrix} I & T_{11}^{-1} T_{12} & T_{11}^{-1} b_1 \\ 0 & I & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Both the vector

$$b_S = b_2 - T_{12}^{\mathrm{H}} T_{11}^{-1} b_1$$

and the Schur complement

$$T_S = T_{22} - T_{12}^{\mathrm{H}} T_{11}^{-1} T_{12}$$

can be found from the matrix $S(z)$ returned by `bsfschur`(). In fact, the generators of the matrix

$$\begin{bmatrix} T_S & b_S \\ b_S^{\mathrm{H}} & 2 - \frac{1}{\delta_{n/2}} \end{bmatrix}$$

are available in polynomial form as the first row of $S_{n/2}^{(n/2)}(z)$. The Schur complements of the augmented matrix, stored in $S(z)$, are the data that will be used to solve $Tx = b$.

Given a linear system partitioned as

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{12}^{\mathrm{H}} & T_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

elimination gives

$$\begin{bmatrix} T_{11} & T_{12} \\ 0 & T_S \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_S \end{bmatrix}.$$

Block back-substitution gives two smaller linear systems

$$T_S x_2 = b_S, \qquad T_{11} x_1 = b_1 - T_{12} x_2. \tag{22}$$

Using `bsfschur`() the computation of generators for the Schur complement system and its right-hand-side is $O(n \log^2(n))$.

To form the right-hand-side for $T_{11} x_1 = b_1 - T_{12} x_2$ we note that T_{12} is a block of the Toeplitz-like matrix T, so that it is also Toeplitz-like. More precisely, if we partition the displacement equation for T, $T - ZTZ^{\mathrm{H}} = vv^{\mathrm{H}} - ww^{\mathrm{H}}$, as

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{12}^{\mathrm{H}} & T_{12} \end{bmatrix} - \begin{bmatrix} Z_{11} & 0 \\ e_1 e_{n/2}^{\mathrm{H}} & Z_{22} \end{bmatrix} \begin{bmatrix} T_{11} & T_{12} \\ T_{12}^{\mathrm{H}} & T_{12} \end{bmatrix} \begin{bmatrix} Z_{11}^{\mathrm{H}} & e_{n/2} e_1^{\mathrm{H}} \\ 0 & Z_{22}^{\mathrm{H}} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \begin{bmatrix} v_1^{\mathrm{H}} & v_2^{\mathrm{H}} \end{bmatrix} - \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \begin{bmatrix} w_1^{\mathrm{H}} & w_2^{\mathrm{H}} \end{bmatrix}$$

then

$$T_{12} - Z_{11}T_{12}Z_{22}^{\mathrm{H}} = v_1 v_2^{\mathrm{H}} - w_1 w_2^{\mathrm{H}} + Z_{11}T_{11}e_{n/2}e_1^{\mathrm{H}}. \tag{23}$$

Thus T_{12} is in general a displacement rank 3 Toeplitz-like matrix. Multiplication by T_{12} is $O(n\log(n))$ using the FFT. Thus both half-size systems (22) can be computed efficiently.

This motivates the following divide-and-conquer algorithm. We assume that the inputs $S(z)$ and $P(z)$ are partitioned as in (19) and (20).

```
function $x$=solve($S(z)$,$P(z)$,$n$)
        if $n > 1$ then
          $x_2$=solve ($S_{n/2}^{(n/2)}(z)$,$P_{n/2}^{(n/2)}(z)$,$n/2$)
          $b_1 = \mathtt{vec}(\beta(z)/\delta)$
          $b_1 = b_1(1:n/2)$
          $T = \mathtt{toeplitz}(v(z), w(z))$
          $T_{12} = T(1:n/2, n/2+1:n)$
          $b_1 = b_1 - T_{12}x_2$
          $\beta(z) = \begin{bmatrix} 1 & z & \cdots z^{n/2-1} \end{bmatrix} b_1$
          $[c(z),\ d(z),\ g,\ S_{n/2}^{(0)}(z)]$=
              rhs($S_{n/2}^{(0)}(z)$, $P_{n/2}^{(0)}(z)$,$\beta^{(n/2)}(z)$, 1, $n/2$)
          $x_1$=solve ($S_{n/2}^{(0)}(z)$,$P_{n/2}^{(0)}(z)$,$n/2$)
          $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
        else
          $x = (v(z)\overline{v(z)} - w(z)\overline{w(z)})^{-1}\beta(z)/\delta$
        endif
```

The function `toeplitz`() constructs a Toeplitz-like matrix from the generators $v_0(z)$ and $w_0(z)$. The function `vec`() forms a vector from the coefficients of a polynomial. These functions make possible the matrix notation

$$b_1 - T_{12}x_2.$$

In practice, this would not be done explicitly; instead the multiplication by T_{12} would be carried out with the generators and the FFT using only $O(n\log(n))$ operations. Unfortunately the call to `rhs`() is not so efficient; it is $O(n\log^2(n))$. As we will see, this makes the procedure `solve`() an $O(n\log^3(n))$ algorithm.

In the case that the recursion terminates with $n = 1$ the matrix T is 1×1, $v(z)$, $w(z)$ and $\beta(z)$ are constants and

$$T = v(z)\overline{v(z)} - w(z)\overline{w(z)}, \qquad b = \beta(z)/\delta$$

so that the solution to $Tx = b$ is

$$x = (v(z)\overline{v(z)} - w(z)\overline{w(z)})^{-1}\beta(z)/\delta.$$

The function assumes that $S(z)$ and $P(z)$ are available. Thus `solve`() would be used as follows:

```
$[a(z),\ b(z),\ c(z),\ d(z),\ g,\ S(z),\ P(z)]$= bsfschur($v(z)$, $w(z)$,$\beta(z)$, 1, $k$)
$x$=solve($S(z)$,$P(z)$,$n$)
```

This naturally assumes that the function `rhs()` is also available.

4 Computational Complexity

The complexity of solving a Toeplitz system using `solve()` is $O(n\log^3(n))$. In this section we will provide a more detailed count of arithmetic operations. We will assume that T and b are real and we will use the assumptions from [2] on the complexity of convolutions, in particular that convolutions are implemented using a split-radix FFT so that a real cyclic convolution of two length n vectors takes

$$6n\log_2(n) - 9n + 14$$

real floating point operations.

The cyclic convolution is

$$w_j = \sum_{k=0}^{n-1} x_k y_{j-k}$$

where y_k and x_k are defined for $k = 0, 1, \ldots, n-1$, $y_{-k} = y_{n-k}$ and $j = 0, 1, \ldots, n-1$. Multiplication of polynomials is a linear rather than a cyclic convolution: It is assumed that $y_k = 0$ for $k < 0$ and $j = 0, 1, \cdots, 2n-2$. Thus to multiply two length n polynomials with coefficients given as elements of the vectors $x = [x_k]$ and $y = [y_k]$ we can pad the vectors with n zeros and compute the cyclic convolution

$$w = \begin{bmatrix} x \\ 0 \end{bmatrix} * \begin{bmatrix} y \\ 0 \end{bmatrix}.$$

In analyzing the complexity of `bsfschur()` it is important to take note of the length of each of the convolutions. The equations

$$v_{n/2}(z) = v(z)a_{n/2}^{(0)}(z) + w(z)\tilde{b}_{n/2}^{(0)}(z) \qquad w_{n/2}(z) = v(z)b_{n/2}^{(0)}(z) + w(z)\tilde{a}_{n/2}^{(0)}(z)$$

involve four convolutions, each with one length n vector and one length $n/2$ vector. This can be implemented using a length $3n/2$ cyclic convolution. However, because $v_{n/2}(z)$ and $w_{n/2}(z)$ will be truncated and will have leading zeros, only the middle $n/2$ elements of the convolutions will be needed. It follows that these can be done using four length n convolutions, [2]. This also applies to the convolutions in

$$\beta_{n/2}(z) = v(z)c_{n/2}^{(0)}(z) + w(z)d_{n/2}^{(0)}(z) + \beta(z)g_{n/2}^{(0)}.$$

The convolutions in the equations for $a(z)$, $b(z)$, $c(z)$ and $d(z)$ can also be implemented using length n cyclic convolutions.

Let $B(n)$ be the complexity of `bsfschur()`. The function calls itself twice on half-size problems and performs 14 length n cyclic convolutions. The complexity satisfies

$$B(n) = 2B(n/2) + 14\left(6n\log_2(n) - 9n + 14\right) + \frac{19}{2}n + 2$$

or

$$B(n) = 2B(n/2) + 84n\log_2(n) - \frac{233}{2}n + 198. \tag{24}$$

The term $19n/2 + 2$ in the first expression is the cost of the vector adds, scalar-vector multiplies and two scalar-scalar multiplies. The scalar multiplies are the products $g_{n/2}^{(0)}g_{n/2}^{(n/2)}$ and $\delta g_{n/2}^{(0)}$. In assessing the cost of the vector additions, we have assumed that elements of vectors that are to be truncated are not computed.

The general solution to (24) is

$$B(n) = 42n\log_2^2(n) - \frac{149}{2}n\log_2(n) + Cn - 198.$$

To determine the constant C we note that if $n = 1$ then `bsfschur`() performs 18 real operations. Thus

$$B(1) = C - 198 = 18$$

so that

$$B(n) = 42n\log_2^2(n) - \frac{149}{2}n\log_2(n) + 216n - 198.$$

Now let the complexity of `rhs`() be $R(n)$. Only six convolutions are required so

$$R(n) = 2R(n/2) + 6\left(6n\log_2(n) - 9n + 14\right) + \frac{13}{2}n + 2.$$

or

$$R(n) = 2R(n/2) + 36n\log_2(n) - \frac{95}{2}n + 86.$$

The general solution to this is

$$R(n) = 18n\log_2^2(n) - \frac{59}{2}n\log_2(n) + Cn - 86.$$

Since $R(1) = C - 86 = 16$

$$R(n) = 18n\log_2^2(n) - \frac{59}{2}n\log_2(n) + 102n - 86.$$

For `solve`() we assume that the multiplication by the $n/2 \times n/2$ Toeplitz-like matrix T_{12} involves

$$4\left(6(2n)\log_2(2n) - 9(2n) + 14\right) + n/2 = 48n\log_2(n) - \frac{47}{2}n + 56$$

operations. The justification is as follows. We note that

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{12}^{\mathrm{H}} & T_{22} \end{bmatrix} \begin{bmatrix} 0 \\ x_2 \end{bmatrix} = \begin{bmatrix} T_{12}x_2 \\ T_{22}x_2 \end{bmatrix}$$

so that to multiply a vector by T_{12} efficiently it suffices to have an efficient means of multiplying a vector by T. Since T is Toeplitz-like it can be represented through the Gohberg-Semencul formula

$$T = L_+L_+^{\mathrm{H}} - L_-L_-^{\mathrm{H}}$$

where $L_\pm$ are lower triangular and Toeplitz. Each matrix $L\pm$ can then be embedded in a circulant matrix of size $2n \times 2n$. Multiplication by these circulants is simply a cyclic convolution. Thus multiplication by T can be reduced to 4 convolutions of size $2n$ with cost

$$4\left(6(2n)\log_2(2n) - 9(2n) + 14\right).$$

The additional $n/2$ is the complexity of the length $n/2$ add that finally computes $T_{12}x_2$. Alternately it is possible to use (23) directly and in so doing avoid expanding the size of the convolutions by a factor of 4. However, this would involve a greater number of convolutions and the additional computation of $T_{11}e_n$.

Under the above assumption, if $S(n)$ is the cost of `solve`() then

$$S(n) = 2S(n/2) + R(n/2) + (48n\log_2(n) - \frac{47}{2}n + 56) + n/2$$

or

$$S(n) = 2S(n/2) + 9n\log_2^2(n) + \frac{61}{4}n\log_2(n) + \frac{207}{4}n - 30.$$

The general solution is

$$S(n) = 3n\log_2^3(n) + \frac{97}{8}n\log_2^2(n) + \frac{487}{8}n\log_2(n) + Cn + 30.$$

For the case $n = 1$, $S(1) = C + 30 = 4$ so that

$$S(n) = 3n\log_2^3(n) + \frac{97}{8}n\log_2^2(n) + \frac{487}{8}n\log_2(n) - 26n + 30.$$

real floating point operations.

To fully solve a Toeplitz system requires an initial call to `bsfschur`() so that the complexity of solving $Tx = b$ is

$$T(n) = S(n) + B(n) = 3n\log_2^3(n) + \frac{433}{8}n\log_2^2(n) - \frac{109}{8}n\log_2(n) + 190n - 168. \tag{25}$$

To compare this superfast algorithm to the most comparable fast methods, we note that Schur's algorithm requires $3n^2$ operations to compute the Cholesky factor of T. Another $2n^2$ is required for back-substitution, so that the solution of $Tx = b$ requires roughly $5n^2$ operations. The smallest value of n for which $T(n)$ is smaller than $5n^2$ is $n = 2148$. Of course `solve`() assumes that n is a power of two and the smallest power of 2 for which `solve`() has a smaller operation count than the Schur algorithm is $n = 4096$. Nevertheless the algorithm is very close to breaking even at $n = 2048$.

In contrast, since the superfast Schur algorithm of [2] computes generators of T^{-1}, it is perhaps most naturally compared to the Levinson algorithm, which also computes these generators. In [2] it was shown that the superfast Schur algorithm breaks even in comparison to the Levinson algorithm for $n = 256$.

Finally, we note that the overall storage required by the recursive algorithm is $O(n\log(n))$. Instead of making assumptions about how computer memory is used, we will analyze the storage required by $S(z)$. All the other polynomials computed by the algorithm could be stored in a similar array so that is sufficient to show that $S(z)$ requires $O(n\log(n))$ storage. Note that this analysis assumes the recursive formulation given here; there is redundancy in $S(z)$ and it might be possible to develop a non-recursive algorithm that uses only $O(n)$ storage.

Given an $n \times n$ Toeplitz system, let $M(n)$ be the storage required for $S_0^{(0)}(z)$. Then

$$M(n) = 2M(n/2) + 3n$$

which has solution

$$M(n) = 3n\log_2(n) + Cn.$$

If $n = 1$ then $S(z)$ only stores 3 constants so that

$$M(1) = C = 3$$

so that

$$M(n) = 3n\log_2(n) + 3n.$$

Experiment	$\kappa(T)$	$\|T_{11}^{-1}T_{12}\|$	`solve`()	Inversion
1	5.14×10^8	25.3	1.2×10^{-14}	6.6×10^{-10}
	3.5×10^8	12.0	6.7×10^{-16}	2.2×10^{-10}
	2.4×10^9	25.7	9.1×10^{-15}	1.02×10^{-8}
2	1.1×10^{14}	37.8	9.4×10^{-15}	1×10^{-4}
	2.3×10^{13}	11.6	5.0×10^{-15}	7.5×10^{-4}
	1.5×10^{14}	83.6	6.42×10^{-15}	8.7×10^{-5}
3	7.8×10^{11}	1.1×10^4	2.0×10^{-10}	2.5×10^{-7}
	3.15×10^{13}	1.7×10^3	2.7×10^{-13}	3.6×10^{-6}

Table 1: Relative residuals

5 Numerical Experiments

The reason for formulating a superfast algorithm in terms of factorization and divide-and-conquer back-substitution was in the hope of achieving some of the stability inherent in unstructured triangularization and back-substitution. Unfortunately the algorithm is quite complicated; a rigorous error analysis has not been performed and might well be extremely difficult. Instead we will attempt to assess stability through numerical experiments. All experiments were conducted using code written in Matlab and run on a Pentium III PC with machine precision approximately $\epsilon = 1 \times 10^{-16}$. The FFT routines used were those built into Matlab.

For the first experiment we generated a 128×128 positive definite Toeplitz matrix from random Schur parameters ρ_k distributed uniformly over the interval $[-.5, .5]$. These parameters resulted in ill-conditioned but numerically nonsingular matrices. For the right hand side vector b we randomly generated a vector $\hat{x}$ and then formed the product $T\hat{x} = b$. For solutions x obtained by `solve`() and by the superfast inversion algorithm of [2] combined with the Gohberg-Semencul formula for multiplication by T^{-1} the relative residuals

$$r(T, b, x) = \frac{\|Tx - b\|}{\|T\|\|x\| + \|b\|}$$

are shown in the first three lines of Table 1. (Note: each line in the table corresponds to a different matrix). As expected for a method based on inversion, the residuals for the Gohberg-Semencul approach are large. The residuals for `solve`() are what might be expected for a backward stable algorithm. These results were typical for random problems generated in this way.

Next we generated ill-conditioned Toeplitz matrices for which $|\rho_k|$ was close to 1 for some k. In particular, we generated random 128×128 Toeplitz matrices with ρ_k uniformly distributed over $[-.3, .3]$ with two of the ρ_k changed to

$$\rho_{10} = .9999999, \qquad \rho_{15} = -.99.$$

The right hand side vectors were generated in the same manner as before. The results are shown in lines 4–6 of Table 1. Note that these matrices are almost numerically singular. The errors for `solve`() remain on the order of the machine precision while those for the other algorithm have increased with the increasing condition number.

Finally, we devise an experiment to highlight a notable weakness of the new algorithm: `solve`() can lose accuracy when the quantity $\|T_{11}^{-1}T_{12}\|$ (or the equivalent quantity for any of the Schur complements of T) becomes large. It was shown in [14] that for a positive definite Toeplitz matrix, or for the Schur complement of a positive definite Toeplitz matrix, the quantity $\|T_{11}^{-1}T_{12}\|_2$ can be bounded by an expression that depends only on the sizes of the matrices and not on $\|T_{11}^{-1}\|$. The reason is that if

$$\begin{bmatrix} T_{11} & T_{12} \\ T_{12}^{\mathrm{H}} & T_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} L_{11}^{\mathrm{H}} & L_{21}^{\mathrm{H}} \\ 0 & L_{22}^{\mathrm{H}} \end{bmatrix}$$

is an LDL^{H} factorization of T then

$$T_{11}^{-1}T_{12} = (L_{11}D_1L_{11}^{\mathrm{H}})^{-1}L_{11}D_1L_{21}^{\mathrm{H}} = L_{11}^{-1}L_{12}.$$

However it can be shown that the rows of L^{-1} are the coefficients of optimal filters solving a linear prediction problem, [9]. Consequently they have the well-known minimum phase property: the polynomials with coefficients taken from the rows of L^{-1} have zeros only in the unit circle. This implies that

$$|L^{-1}| < \begin{bmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ 1 & 2 & 1 & & & \\ 1 & 3 & 3 & 1 & & \\ 1 & 4 & 6 & 4 & 1 & \\ \vdots & & & & & \ddots \end{bmatrix}.$$

Equivalently

$$|L_{ij}| \le \begin{pmatrix} i-1 \\ j-1 \end{pmatrix}.$$

It is shown in [14] that the same bounds hold for $|L|$ although this fact follows for different reasons; the polynomials formed from the rows of L do not have the minimum phase property and can have zeros outside the unit circle. The result of the two inequalities is that $L_{11}^{-1}L_{12}$, and hence $T_{11}^{-1}T_{12}$, must be bounded by a function of n independent of the size of $\|T_{11}^{-1}\|$. Unfortunately the bounds are not completely satisfcatory: binomial coefficients grow quickly with increasing n. Further, there is an example of a sequence of positive definite Toeplitz matrices for which both L and L^{-1} approach these bounds in the limit. (In this limit the Toeplitz matrix also becomes singular.)

Nevertheless, the bounds are in practice very pessimistic. It is extremely difficult to generate Toeplitz matrices of any reasonable size that come close to achieving these bounds and are still numerically positive definite. For the next experiment we will apply `solve`() to positive definite Toeplitz matrices for which $\|T_{11}^{-1}T_{12}\|$ is as large as we were able to make it. The examples are small because larger examples that are numerically nonsingular could not be generated.

We started with a Toeplitz matrix for which

$$\rho_1 = \rho_2 = \cdots = \rho_7 = .99$$

and

$$\rho_8 = \rho_9 = \cdots = \rho_{32} = .2.$$

The right hand side was generated in the same way as before. The results are on the seventh line of Table 1. Clearly $\|T_{11}^{-1}T_{12}\|$ is larger than before and there has been a proportional increase in the errors.

Nevertheless, it seems that the algorithm is stable in most circumstances. This example is very extreme and even seemingly minor changes in the parameters considerably reduce $\|T_{11}^{-1}T_{12}\|$. Suppose we keep the previous set of Schur parameters, only changing $\rho_4 = .1$ and $\rho_7 = -.8$. The results are on the final line of Table 1. The quantity $\|T_{11}^{-1}T_{12}\|$ has dropped an order of magnitude and the error has improved for `solve`() but not for the inversion. Note that the condition number has become worse; growth in errors is apparently not linked to ill-conditioning in a simple or direct way.

6 Observations

The algorithm proposed in this paper is a divide-and-conquer $O(n \log^3(n))$ method for the solution of positive definite Toeplitz systems. It achieves a crossover point at which it beats the Schur algorithm for $n = 4096$. Its strength over previous superfast methods is that it is observed to be relatively numerically stable. Experiments suggest that this stability is connected with the tendency of the block eliminators

$$\begin{bmatrix} I & 0 \\ -T_{12}^{\mathrm{H}} T_{11}^{-1} & I \end{bmatrix}.$$

to be of modest size when T is positive definite and Toeplitz. It was shown in [14] that the Schur complements of a Toeplitz matrix are insensitive to perturbations when $T_{11}^{-1} T_{12}$ is not large. This might make possible an error analysis based on forward accuracy in computed Schur complements. This is a possible direction for further research.

A stabilized superfast algorithm for nonsymmetric Toeplitz systems was published in [3]. However that algorithm depended in part on iterative refinement for its stability. Iterative refinement is well known to stabilize algorithms that are not too unstable applied to problems that are not too ill-conditioned, [13, 10]. The algorithm of [3] appeared to be stable in most cases, but displayed growth in relative residuals, despite iterative refinement, when tested on some very large problems. In contrast, the algorithm presented here is stable on at least some extremely ill-conditioned problems. Further, the mild degree of instability exhibited by the algorithm is not so extreme as to prevent iterative refinement from restoring backward stability. Unfortunately it is not clear how the new algorithm might be extended to the nonsymmetric Toeplitz matrices considered in [3] or to any broader class of structured matrices. Further it seems likely that the stability of the algorithm depends on $\|T_{11}^{-1} T_{12}\|$ not being too large. Bounds of this sort have been established only for positive definite Toeplitz matrices.

References

[1] G. S. Ammar and W. B. Gragg, *The implementation and use of the generalized Schur algorithm*, in Computational and Combinatorial Methods in Systems Theory, C. I. Byrnes and A. Linquist, eds., North Holland, Amsterdam, 1986, pp. 265–279.

[2] ——, *Superfast solution of real positive definite toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[3] M. V. Barel, G. Heinig, and P. Kravanja, *A stabilized superfast solver for nonsymmetric toeplitz systems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 494–510.

[4] R. R. Bitmead and D. O. Anderson, *Asymptotically fast solution of toeplitz and related systems*, Linear Algebra and Its Applications, 34 (1980), pp. 103–116.

[5] A. Bojanczyk, R. P. Brent, F. R. de Hoog, and D. R. Sweet, *On the stability of the Bareiss and related Toeplitz factorization algorithms*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 40–58.

[6] R. P. Brent, F. Gustavson, and D. Yun, *Fast solution of Toeplitz systems of equations and computation of padé approximants*, J. Algorithms, 1 (1980), pp. 259–295.

[7] S. Chandrasekaran and A. H. Sayed, *Stabilizing the fast generalized Schur algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 950–983.

[8] F. de Hoog, *A new algorithm for solving toeplitz systems of equations*, Linear Algebra and Its Applications, 88/89 (1987), pp. 123–138.

[9] S. Haykin, *Adaptive Filter Theory*, Prentice–Hall, Englewood Cliffs, NJ, 1991.

[10] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, 1996.

[11] B. R. Musicus, *Levinson and fast choleski algorithms for Toeplitz and almost Toeplitz matrices*, research report, Research Lab. of Electronics, Massachusetts Institute of Technology, 1984.

[12] C. Rader and A. Steinhardt, *Hyperbolic Householder transforms*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 269–290.

[13] R. D. Skeel, *Iterative refinement implies numerical stability for Gaussian elimination*, Mathematics of Computation, 35 (1980), pp. 817–832.

[14] M. Stewart, *Cholesky factorization of semidefinite toeplitz matrices*, Linear Algebra and Its Applications, 254 (1997), pp. 497–525.

[15] M. Stewart and G. W. Stewart, *On hyperbolic triangularization: Stability and pivoting*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 847–860.

[16] M. Stewart and P. Van Dooren, *Stability issues in the factorization of structured matrices*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 104–118.