Fast Stable Solvers for Sequentially Semi-separable Linear Systems of Equations

S. Chandrasekaran[*] P. Dewilde M. Gu[†] T. Pals[*] A.-J. van der Veen

January 14, 2003

Abstract

We define the class of sequentially semi-separable matrices in this paper. Essentially this is the class of matrices which have low numerical rank on their off diagonal blocks. Examples include banded matrices, semi-separable matrices, their sums as well as inverses of these sums. Fast and stable algorithms for solving linear systems of equations involving such matrices and computing Moore-Penrose inverses are presented. Supporting numerical results are also presented. In addition, fast algorithms to construct and update this matrix structure for any given matrix are presented. Finally, numerical results that show that the coefficient matrices resulting from global spectral discretizations of certain integral equations indeed have this matrix structure are given.

Keywords: Backward stability, fast direct solver, fast multi-pole method, integral equations, method of moments, scattering theory, semi-separable matrices, spectral methods, Moore-Penrose inverses.

1 Introduction

In this paper we will present fast backward stable algorithms for the solution (or in the case of singular or rectangular systems the determination of Moore-Penrose inverses) of a large class of structured matrices which can be either sparse or dense. The structure concerned is called 'semi-separable' and is a matrix analog of semi-separable integral kernels as described by Kailath in [29]. This matrix analog was most likely first described by Gohberg, Kailath and Koltracht in [20]. In that paper it is shown that, under further technical restrictions, an LDU factorization is possible with a complexity n^2N where n is the complexity of the semi-separable description and N the dimension of the matrix - in effect an algorithm linear in the size of the matrix, when n is small. In a number of papers Alpay, Dewilde and Dym introduce a new formalism for time-varying systems which provides for a framework closely analog to the classical time invariant state space description and which allows for the generalization of many time invariant methods to the time-varying case [1, 2]. When applied to matrices, this formalism generalizes the formalism used in [20] and allows for more general types of efficient operations (by 'efficient' we mean operations that are linear in the size of the matrix). In the book *Time-varying Systems and Computations [14]*, Dewilde and van der

[*]Partially supported by grant B521537 from Lawrence Livermore national Laboratory and NSF Career Award CCR-9734290. Email: shiv@ece.ucsb.edu.

[†]This research was supported in part by NSF Career Award CCR-9702866 and by Alfred Sloan Research Fellowship BR-3720. Email: mgu@math.berkeley.edu.

Matrix	U_i	V_i	W_i	P_i	Q_i	R_i
Dimensions	$m_i \times k_i$	$m_i \times k_{i-1}$	$k_{i-1} \times k_i$	$m_i \times l_i$	$m_i \times l_{i+1}$	$l_{i+1} \times l_i$

Table 1: Dimensions of matrices in (1). k_i and l_i are column dimensions of U_i and P_i, respectively.

Veen describe the various operations that are possible on time-varying systems in great detail, including the efficient application of orthogonal transformations. In particular, they show how a URV type transformation on a general, (possibly infinite dimensional) semi-separable system can be done with an efficient recursive procedure. This procedure is based on the ideas presented in [39] and then further elaborated by Dewilde and van der Veen in [15] and by Eidelman and Gohberg in [19]. In the former paper the connection with Kalman filtering as a special case of the procedures is also discussed.

In this paper we will present an implicit version of these algorithms for finite-dimensional matrices. This essentially combines the fast solution techniques for banded plus semi-separable linear systems of equations of Chandrasekaran and Gu [5] with related techniques of Dewilde and van der Veen for time-varying systems [14]. In addition, we indicate new improvements on these methods, and in particular a method to compute Moore-Penrose inverses of general semi-separable matrices with a single top-down recursion.

We will also use the proposed techniques to suggest fast direct solvers for two classes of spectral methods for which there had been no known fast direct solvers (not even unstable ones). This will illustrate the usefulness of the algorithms presented in this paper. One spectral method is due to Greengard and Rokhlin [23, 31, 37, 38] for two-point boundary-value problems, and the other is by Kress [12] for solving the integral equations of classical exterior scattering theory in two dimensions.

To be more specific, let A be an $N \times N$ (possibly complex) matrix satisfying the matrix structure. Then there exist n positive integers $m_1, \cdots, m_n$ with $N = m_1 + \cdots + m_n$ to block-partition A as

$$A = (A_{i,j}), \quad \text{where } A_{ij} \in \mathbf{C}^{m_i \times m_j} \text{ satisfies } A_{ij} = \begin{cases} D_i, & \text{if } i = j, \\ U_i W_{i+1} \cdots W_{j-1} V_j^H, & \text{if } j > i, \\ P_i R_{i-1} \cdots R_{j+1} Q_j^H, & \text{if } j < i. \end{cases} \tag{1}$$

Here we use the superscript H to denote the Hermitian transpose. The sequences $\{U_i\}_{i=1}^{n-1}$, $\{V_i\}_{i=2}^{n}$, $\{W_i\}_{i=2}^{n-1}$, $\{P_i\}_{i=2}^{n}$, $\{Q_i\}_{i=1}^{n-1}$, $\{R_i\}_{i=2}^{n-1}$ and $\{D_i\}_{i=1}^{n}$ are all matrices whose dimensions are defined in Table 1. While any matrix can be represented in this form for large enough k_i's and l_i's, our main focus will be on matrices of this special form that have relatively small values for the k_i's and l_i's (see Section 6). In the above equation, empty products are defined to be the identity matrix. For $n = 4$, the matrix A has the form

$$A = \begin{pmatrix} D_1 & U_1 V_2^H & U_1 W_2 V_3^H & U_1 W_2 W_3 V_4^H \\ P_2 Q_1^H & D_2 & U_2 V_3^H & U_2 W_3 V_4^H \\ P_3 R_2 Q_1^H & P_3 Q_2^H & D_3 & U_3 V_4^H \\ P_4 R_3 R_2 Q_1^H & P_4 R_3 Q_2^H & P_4 Q_3^H & D_4 \end{pmatrix}.$$

We say that the matrix A is **sequentially semi-separable** if it satisfies (1). In the case where all W_i and R_i are identities, A reduces to a block-diagonal plus semi-separable matrix, which can be handled directly using techniques in Chandrasekaran and Gu [5]. It is shown in [14] that this class of matrices is closed under inversion and includes banded matrices, semi-separable matrices as well as their inverses as special cases.

It should be noted that the sequentially semi-separable structure of a given matrix A depends on the sequence m_i. Different sequences will lead to different representations. Through out the first part of this paper we will assume that the D_i's are square matrices, we generalize to Moore-Penrose inverses in later sections.

Our main objective of this paper is to present a fast backward stable algorithm for solving systems of linear equations where the coefficient matrix satisfies (1). Additionally, we demonstrate the usefulness of this approach by showing how this algorithm can be used to potentially speed-up a two-dimensional exterior Helmholtz solver and also greatly reduces the amount of memory needed.

In Section 2, we discuss a fast algorithm to multiply the matrix in (1) by any given vector or matrix; such an algorithm is already presented in [14], but we include it here for completeness and convenience. In Section 5, we present a fast backward stable solver for solving sequentially semi-separable linear systems of equations; in Sections 6 and 8, we show how to effectively construct and update the sequentially semi-separable representation for a broad class of dense matrices, thereby significantly reducing the solution time for related linear systems of equations; and in Section 12, we discuss applications of the above algorithms to solve equations arising from the spectral discretization of two-point boundary-value problems and exterior scattering problems.

2 Fast Multiplication

We begin by presenting the fast multiplication algorithm which is useful for efficiently checking the backward error of the solution generated by the fast direct solver.

Assume that we want to multiply the sequentially semi-separable matrix A with the vector x to get $b = Ax$. We partition $x = (\,x_i\,)$ and $b = (\,b_j\,)$ such that x_i and b_i have m_i rows. Then

$$\begin{aligned} b_j \;=\;& P_j R_{j-1}\cdots R_2 Q_1^H x_1 + P_j R_{j-1}\cdots R_3 Q_2^H x_2 + \cdots + P_j Q_{j-1}^H x_{j-1} + D_j x_j \\ +\;& U_j V_{j+1}^H x_{j+1} + U_j W_{j+1} V_{j+2}^H x_{j+2} + \cdots + U_j W_{j+1}\cdots W_{n-1} V_n^H x_n. \end{aligned}$$

Staring at this formula we realize that we can speed up these calculations by first calculating the intermediate quantities:

$$\begin{aligned} g_n &= V_n^H x_n, \quad \text{and} \quad g_i = V_i^H x_i + W_i g_{i+1}, \qquad i = n-1, \ldots, 1; \\ h_1 &= Q_1^H x_1, \quad \text{and} \quad h_j = Q_j^H x_j + R_j h_{j-1}, \qquad j = 2, \ldots, n. \end{aligned}$$

With these quantities we have that

$$b_1 = D_1 x_1 + U_1 g_2, \quad b_i = P_i h_{i-1} + D_i x_i + U_i g_{i+1}, \;\; 1 < i < n, \quad \text{and} \quad b_n = P_n h_{n-1} + D_n x_n.$$

Here is the flop count. It takes at most $2\sum k_{i-1}(m_i + 1 + k_i)$ flops to compute the g_i's, at most $2\sum l_{i+1}(m_i + 1 + l_i)$ flops to compute the h_i's. Similarly, it takes at most $2\sum m_i(m_i + l_i + k_i + 1)$ flops to compute all the b_i's from the g_i's and h_i's. Hence the total number of flops does not exceed

$$2\sum k_{i-1}(m_i + 1 + k_i) + l_{i+1}(m_i + 1 + l_i) + m_i(m_i + l_i + k_i + 1).$$

To see the benefits of this algorithm consider the total number of flops when $m_i = m$ and $k_i = k$ and $l_i = l$ for all i. It works out to be at most $11n(l^2 + k^2 + m^2)$ flops, as opposed to $2n^2m^2$ flops for doing it by the usual multiplication method. As can be seen, if k and l are small compared to $m\sqrt{n}$ substantial savings in flops can be obtained.

This algorithm can be straightforwardly generalized to deal with the case where x is replaced by an $N \times K$ matrix for $K > 1$ with similar flop savings. We note that all the matrix-vector products now become matrix-matrix products, hence the usual benefits of level-3 BLAS still remain [16].

3 Fast SSS matrix-matrix multiplication

The basic algorithms for multiplying either two upper or lower triangular matrices in SSS form is presented in [14]. We now present a fast and stable method to multiply two matrices that have SSS representations. This algorithm is an order of magnitude faster than applying the algorithm in section 2 either column by column, or row by row. Let A and B be matrices in SSS form that are conformally partitioned. We first define forward and backward recursions

$$\begin{array}{rcll} G_1 & = & 0, \quad G_{i+1} = Q_i^H(A)U_i(B) + R_i(A)G_iW_i(B), & , i = 1, \cdots, n-1. \\ H_n & = & 0, \quad H_{i-1} = V_i^H(A)P_i(B) + W_i(A)H_iR_i(B), & , i = n, \cdots, 2. \end{array}$$

Theorem 1 *The SSS form of the matrix $C = AB$ can be computed through the following recursions:*

$$\begin{array}{rcl} D_i(C) & = & D_i(A)D_i(B) + P_i(A)G_iV_i^H(B) + U_i(A)H_iQ_i^H(B), \\ P_i(C) & = & \left(D_i(A)P_i(B) + U_i(A)H_iR_i(B) \quad P_i(A) \right), \\ R_i(C) & = & \begin{pmatrix} R_i(B) & \\ Q_i^H(A)P_i(B) & R_i(A) \end{pmatrix}, \\ Q_i(C) & = & \left(Q_i(B) \quad D_i^H(B)Q_i(A) + V_i(B)G_i^HR_i^H(A) \right), \\ U_i(C) & = & \left(D_i(A)U_i(B) + P_i(A)G_iW_i(B) \quad U_i(A) \right), \\ W_i(C) & = & \begin{pmatrix} W_i(B) & \\ V_i^H(A)U_i(B) & W_i(A) \end{pmatrix}, \\ V_i(C) & = & \left(V_i(B) \quad D_i^H(B)V_i(A) + Q_i(B)H_i^HW_i^H(A) \right). \end{array}$$

Proof. We prove the recursions by a mathematical induction on n, the number of blocks in the SSS representation of C. To avoid redundant work, we postpone the proof on the base step to the end, and begin by assuming that the recursions hold for any A and B with at most $n \geq 3$ blocks.

Now let A and B be SSS matrices with $n+1$ blocks. We turn them and C to SSS matrices with n blocks by merging the last 2 blocks into one in each matrix. The n-block SSS representation is the same as in the $(n+1)$-block representation in the first $n-1$ blocks. In the n-th block, the new representation in C is

$$D_n^{\mathbf{mg}}(C) = \begin{pmatrix} D_n(C) & U_n(C)V_{n+1}^H(C) \\ P_{n+1}(C)Q_n^H(C) & D_{n+1}(C) \end{pmatrix}, \tag{2}$$

$$P_n^{\mathbf{mg}}(C) = \begin{pmatrix} P_n(C) \\ P_{n+1}(C)R_n(C) \end{pmatrix}, \tag{3}$$

$$V_n^{\mathbf{mg}}(C) = \begin{pmatrix} V_n(C) \\ V_{n+1}(C)W_n^H(C) \end{pmatrix}. \tag{4}$$

The new n-th block representations in A and B are similar.

The $(n+1)$-block representation defines the recursion matrices $\{G_i\}_{i=1}^{n+1}$ and $\{H_i\}_{i=1}^{n+1}$. In the n-block representation, the recursion matrices $\{G_i\}_{i=1}^{n}$ remain the same, but there is a slight difference in $\{H_i\}_{i=1}^{n}$. In fact, $H_n^{\mathbf{mg}} = 0$ and

$$\begin{array}{rcl} H_{n-1}^{\mathbf{mg}} & = & (V^{\mathbf{mg}})_n^H(A)P_n^{\mathbf{mg}}(B) \\ & = & \left(V_n^H(C) \quad W_n(C)V_{n+1}^H \right) \begin{pmatrix} P_n(A) \\ P_{n+1}(A)R_n(A) \end{pmatrix} \\ & = & V_n^H(C)P_n(A) + W_n(C)\left(V_{n+1}^HP_{n+1}(A)\right)R_n(A) \\ & = & V_n^H(C)P_n(A) + W_n(C)H_nR_n(A) = H_{n-1}. \end{array}$$

It follows that H_i remains the same for $i = n-1, \cdots, 1$. Hence the recursion formulas remain identical in both representations for any $i \leq n-1$. Consequently, the induction assumption implies that the recursion formulas remain true for any $i \leq n-1$ in the $(n+1)$-block representation. To show that these formulas are also true for $i = n$ and $n+1$ in this representation, we note that the induction assumption on $P_n^{\mathbf{mg}}(C)$ is

$$\begin{aligned}
P_n^{\mathbf{mg}}(C) &= \begin{pmatrix} D_n^{\mathbf{mg}}(A)P_n^{\mathbf{mg}}(B) + U_n^{\mathbf{mg}}(A)H_n^{\mathbf{mg}}R_n^{\mathbf{mg}}(B) & P_n^{\mathbf{mg}}(A) \end{pmatrix} \\
&= \left(\begin{pmatrix} D_n(A) & U_n(A)V_{n+1}^H(A) \\ P_{n+1}(A)Q_n^H(A) & D_{n+1}(A) \end{pmatrix} \begin{pmatrix} P_n(B) \\ P_{n+1}(B)R_n(B) \end{pmatrix} \quad \begin{pmatrix} P_n(A) \\ P_{n+1}(A)R_n(A) \end{pmatrix} \right) \\
&= \begin{pmatrix} D_n(A)P_n(B) + U_n(A)H_nR_n(B) & P_n(A) \\ P_{n+1}(A)Q_n^H(A)P_n(B) + D_{n+1}(A)P_{n+1}(B)R_n(B) & P_{n+1}(A)R_n(A) \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} D_n(A)P_n(B) + U_n(A)H_nR_n(B) & P_n(A) \end{pmatrix} \\ \begin{pmatrix} D_{n+1}(A)P_{n+1}(B) & P_{n+1}(A) \end{pmatrix} \begin{pmatrix} R_n(B) \\ Q_n^H(A)P_n(B) & R_n(B) \end{pmatrix} \end{pmatrix}.
\end{aligned}$$

By comparing this with (3), we set

$$\begin{aligned}
R_n(C) &= \begin{pmatrix} R_n(B) \\ Q_n^H(A)P_n(B) & R_n(B) \end{pmatrix}, \\
P_i(C) &= \begin{pmatrix} D_i(A)P_i(B) + U_i(A)H_iR_i(B) & P_i(A) \end{pmatrix}, \quad i = n, n+1.
\end{aligned}$$

Similarly, the induction assumption on $W_n^{\mathbf{mg}}(C)$ simplifies to

$$\begin{aligned}
V_n^{\mathbf{mg}}(C) &= \begin{pmatrix} V_n^{\mathbf{mg}}(B) & \left(D_n^{\mathbf{mg}}\right)^H(B)V_n^{\mathbf{mg}}(A) + Q_n^{\mathbf{mg}}(B)\left(H_n^{\mathbf{mg}}\right)^H\left(W_n^{\mathbf{mg}}\right)^H(A) \end{pmatrix} \\
&= \left(\begin{pmatrix} V_n(B) \\ V_{n+1}(B)W_n^H(B) \end{pmatrix} \quad \begin{pmatrix} D_n(B) & U_n(B)V_{n+1}^H(B) \\ P_{n+1}(B)Q_n^H(B) & D_{n+1}(B) \end{pmatrix}^H \begin{pmatrix} V_n(A) \\ V_{n+1}(A)W_n^H(A) \end{pmatrix} \right) \\
&= \begin{pmatrix} \begin{pmatrix} V_n(B) & D_n^H(B)V_n(A) + Q_n(B)H_n^HW_n^H(A) \end{pmatrix} \\ \begin{pmatrix} V_{n+1}(B) & D_{n+1}^H(B)V_{n+1}(A) \end{pmatrix} \begin{pmatrix} W_n(B) \\ V_n^H(A)U_n(B) & W_n(A) \end{pmatrix}^H \end{pmatrix}.
\end{aligned}$$

Now equation (4) allows identification

$$\begin{aligned}
W_n(C) &= \begin{pmatrix} W_n(B) \\ V_n^H(A)U_n(B) & W_n(A) \end{pmatrix}, \\
V_i(C) &= \begin{pmatrix} V_i(B) & D_i^H(B)V_i(A) + Q_i(B)H_i^HW_i^H(A) \end{pmatrix}, \quad i = n, n+1.
\end{aligned}$$

Finally, the induction assumption on $D_n^{\mathbf{mg}}(C)$ simplifies to

$$\begin{aligned}
D_n^{\mathbf{mg}}(C) &= D_n^{\mathbf{mg}}(A)D_n^{\mathbf{mg}}(B) + P_n^{\mathbf{mg}}(A)G_n\left(V_n^{\mathbf{mg}}(B)\right)^H + U_n^{\mathbf{mg}}(A)H_n^{\mathbf{mg}}\left(Q_n^{\mathbf{mg}}(B)\right)^H \\
&= \begin{pmatrix} D_n(A) & U_n(A)V_{n+1}^H(A) \\ P_{n+1}(A)Q_n^H(A) & D_{n+1}(A) \end{pmatrix} \begin{pmatrix} D_n(B) & U_n(B)V_{n+1}^H(B) \\ P_{n+1}(B)Q_n^H(B) & D_{n+1}(B) \end{pmatrix} \\
&\quad + \begin{pmatrix} P_n(A) \\ P_{n+1}(A)R_n(A) \end{pmatrix} G_n \left(\begin{pmatrix} V_n(B) \\ V_{n+1}(B)W_n^H(B) \end{pmatrix}^H \right).
\end{aligned}$$

We can further simplify the $(1,2)$ block of $D_n^{\mathbf{mg}}(C)$ to

$$\begin{aligned}
& D_n(A)U_n(B)V_{n+}(B)^H + U_n(A)V_{n+1}(A)^HD_{n+1}(B) + P_n(A)G_nW_n(B)V_{n+1}(B)^H \\
= \; & \begin{pmatrix} D_n(A)U_n(B) + P_n(A)G_nW_n(B) & U_n(A) \end{pmatrix} V_{n+1}(C)^H.
\end{aligned}$$

According to (2), we can now identify

$$U_n(C) = \begin{pmatrix} D_n(A)U_n(B) + P_n(A)G_nW_n(B) & U_n(A) \end{pmatrix}.$$

In addition, by simplifying the other three blocks of $D_n^{\mathbf{mg}}(C)$, we can set

$$\begin{aligned}
Q_n(C) &= \begin{pmatrix} Q_n(B) & D_n^H(B)Q_n(A) + V_n(B)G_n^HR_n^H(A) \end{pmatrix}, \\
D_i(C) &= D_i(A)D_i(B) + P_i(A)G_iV_i^H(B) + U_i(A)H_iQ_i^H(B), \quad i = n, n+1.
\end{aligned}$$

This concludes the induction.

We now complete the recursion by considering the base step. It is straightforward to verify that

$$\begin{pmatrix} D_1(A) & U_1(A)V_2^H(A) \\ P_2(A)Q_1(A)^H & D_2(A) \end{pmatrix} \begin{pmatrix} D_1(B) & U_1(B)V_2^H(B) \\ P_2(B)Q_1(B)^H & D_2(B) \end{pmatrix} = \begin{pmatrix} D_1(C) & U_1(C)V_2^H(C) \\ P_2(C)Q_1(C)^H & D_2(C) \end{pmatrix},$$

where

$$\begin{aligned} P_2(C) &= (\, D_2(A)P_2(B) \quad P_2(A)\,), \\ V_2(C) &= (\, V_2(B) \quad D_2^H(B)V_2(A)\,), \\ Q_1(C) &= (\, Q_1(B) \quad D_1^H(B)Q_1(A) + V_1(B)R_1^H(A)\,), \\ U_1(C) &= (\, D_1(A)U_1(B) + P_1(A)W_1(B) \quad U_1(A)\,), \\ D_i(C) &= D_i(A)D_i(B) + P_i(A)G_iV_i^H(B) + U_i(A)H_iQ_i^H(B), \quad i = 1, 2. \end{aligned}$$

Now repeat the induction step with $n = 2$ with minor modifications, it is straightforward to verify that all the recursive formulas hold for $n = 3$ as well. Hence they must hold for all $n \geq 3$.

3.1 Flop count

We now estimate the flop count for this algorithm when $m_i = p = k_i = l_i$. In this case the total flop count is at most $40np^3$ flops. As can be seen the constant is of reasonable size.

4 Fast Forward and Backward Substitutions

The SSS structure also allows a fast method to compute $L^{-1}B$ and $R^{-1}B$ for lower and upper triangular SSS matrices L and U and a general SSS matrix B. Again, this algorithm is an order of magnitude faster than computing $L^{-1}B$ and $R^{-1}B$ column by column.

We first consider $L^{-1}B$, where L is lower triangular and both L and B are matrices in SSS form that are conformally partitioned. We need the following lemma

Lemma 2 *Assume that the lower triangular SSS matrix*

$$L = \begin{pmatrix} D_1(L) & & \\ P_2(L)Q_1^H(L) & D_2(L) & \\ \vdots & \ddots & \ddots \\ P_n(L)R_{n-1}(L)\cdots R_2(L)Q_1^H(L) & \cdots & D_n(L) \end{pmatrix}$$

is non-singular. Then L^{-1} is a lower triangular SSS matrix with

$$\begin{aligned} D_i\left(L^{-1}\right) &= D_i^{-1}(L), \quad P_i\left(L^{-1}\right) = -D_i^{-1}(L)P_i(L), \\ Q_i\left(L^{-1}\right) &= \left(D_i^{-1}(L)\right)^H Q_i(L), \quad R_i\left(L^{-1}\right) = R_i(L) - Q_i(L)^H D_i^{-1}(L)P_i(L). \end{aligned}$$

According to the lemma above and Theorem 1, the SSS representation of the forward substitution matrix $L^{-1}B$ can be obtained by plugging the SSS representation of L^{-1} into the matrix multiplication recursions of Theorem 1. The following recursions for $L^{-1}B$ are simplified from these recursions by taking advantage of the fact that L^{-1} is lower triangular. We omit the proof.

Define the following intermediate quantities

$$\begin{aligned}\widehat{U}_i(B) &= D_i^{-1}(L)U_i(B), \quad \widehat{P}_i(B) = D_i^{-1}(L)P_i(B), \quad \widehat{D}_i(B) = D_i^{-1}(L)D_i(B),\\ \widehat{P}_i(L) &= D_i^{-1}(L)P_i(L), \quad \widehat{R}_i(L) = R_i(L) - Q_i(L)^H\widehat{P}_i(L).\end{aligned}$$

We also define the forward recursion

$$G_1 = 0, \quad G_{i+1} = Q_i^H(L)\widehat{U}_i(B) + \widehat{R}_i(L)G_iW_i(B), \quad , i = 1, \cdots, n-1.$$

The SSS form of the matrix $L^{-1}B$ can now be computed through the following recursions:

$$\begin{aligned}D_i(L^{-1}B) &= \widehat{D}_i(B) - \widehat{P}_i(L)G_iV_i^H(B),\\ P_i(L^{-1}B) &= \begin{pmatrix}\widehat{P}_i(B) & -\widehat{P}_i(L)\end{pmatrix},\\ R_i(L^{-1}B) &= \begin{pmatrix} R_i(B) & \\ Q_i^H(L)\widehat{P}_i(B) & \widehat{R}_i(L)\end{pmatrix},\\ Q_i(L^{-1}B) &= \begin{pmatrix}Q_i(B) & \widehat{D}_i^H(B)Q_i(L) + V_i(B)G_i^H\widehat{R}_i(L)\end{pmatrix},\\ U_i(L^{-1}B) &= \widehat{U}_i(B) - \widehat{P}_i(L)G_iW_i(B),\\ W_i(L^{-1}B) &= W_i(B),\\ V_i(L^{-1}B) &= V_i(B).\end{aligned}$$

We now consider $R^{-1}B$, where R is upper triangular and both R and B are matrices in SSS form that are conformally partitioned. Define the following intermediate quantities

$$\begin{aligned}\widehat{U}_i(B) &= D_i^{-1}(R)U_i(B), \quad \widehat{P}_i(B) = D_i^{-1}(R)P_i(B), \quad \widehat{D}_i(B) = D_i^{-1}(R)D_i(B),\\ \widehat{U}_i(R) &= D_i^{-1}(R)P_i(R), \quad \widehat{W}_i(R) = W_i(R) - V_i(R)^H\widehat{U}_i(R).\end{aligned}$$

We also define the backward recursion

$$H_n = 0, \quad H_{i-1} = -V_i^H(R)\widehat{P}_i(B) + \widehat{W}_i(R)H_iR_i(B), \quad , i = n, \cdots, 2.$$

The SSS form of the matrix $R^{-1}B$ now takes the following recursive form:

$$\begin{aligned}D_i(R^{-1}B) &= \widehat{D}_i(B) + \widehat{U}_i(R)H_iQ_i^H(B),\\ P_i(R^{-1}B) &= \widehat{P}_i(B) + \widehat{U}_i(R)H_iR_i(B),\\ R_i(R^{-1}B) &= R_i(B),\\ Q_i(R^{-1}B) &= Q_i(B),\\ U_i(R^{-1}B) &= \begin{pmatrix}\widehat{U}_i(B) & \widehat{U}_i(R)\end{pmatrix},\\ W_i(R^{-1}B) &= \begin{pmatrix} W_i(B) & \\ V_i^H(R)\widehat{U}_i(B) & \widehat{W}_i(R)\end{pmatrix},\\ V_i(R^{-1}B) &= \begin{pmatrix}V_i(B) & -\widehat{D}_i^H(B)V_i(R) + Q_i(B)H_i^H\widehat{W}_i^H(R)\end{pmatrix}.\end{aligned}$$

5 Fast Backward Stable Solver

In this section we describe a recursive and fast backward stable solver for the linear system of equations $Ax = b$, where A satisfies (1) and b itself is an unstructured matrix. But before we present the formal algorithm we demonstrate the key ideas on a 4×4 block matrix example.

5.1 A 4×4 Example

Let the initial equations $Ax = b$ be partitioned as follows:

$$\begin{pmatrix} D_1 & U_1V_2^H & U_1W_2V_3^H & U_1W_2W_3V_4^H \\ P_2Q_1^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ P_3R_2Q_1^H & P_3Q_2^H & D_3 & U_3V_4^H \\ P_4R_3R_2Q_1^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} - \begin{pmatrix} 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \end{pmatrix} \tau, \tag{5}$$

where $\tau = 0$. We have added the extra (zero) vector in the right hand side of (5) to bring the equations into the general form of our recursion.

The algorithm takes one of two essential forms depending on whether k_1, the number of columns of U_1, is strictly less than m_1, the number of rows (columns) of D_1, or not.

5.1.1 Case of $n > 1$ and $k_1 < m_1$: Elimination

Our goal is to first use a (unitary) transformation from the left of A to zero out most of the entries in the first $m_1 - k_1$ rows; and then use a (unitary) transformation from the right of A to produce a $(m_1 - k_1) \times (m_1 - k_1)$ lower-triangular system at the top right corner of A.

To this end, we first apply a unitary transformation q_1^H, from the left of A to the first m_1 rows of the equation such that the first $m_1 - k_1$ rows of U_1 become zero. More specifically let

$$q_1^H U_1 = \begin{array}{cc} \begin{pmatrix} 0 \\ \hat{U}_1 \end{pmatrix} & \begin{array}{c} m_1 - k_1 \\ k_1 \end{array} \end{array}.$$

Now we multiply the first m_1 rows of the equation from the left of A by q_1^H to obtain

$$\left(\begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} A \right) x = \left(\begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} b \right) - \begin{pmatrix} 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \end{pmatrix} \tau.$$

In the new coefficient matrix $\left(\begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} A \right)$, the matrix U_1 has become $\begin{pmatrix} 0 \\ \hat{U}_1 \end{pmatrix}$ and D_1 has become $q_1^H D_1$. Hence the first $m_1 - k_1$ rows of the new matrix are zero starting from the $(m_1 + 1)$-st column. The only difference between $\left(\begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} b \right)$ and b is in b_1, which becomes $q_1^H b_1$. There is no need to multiply the term involving τ since its first block is a zero matrix.

Next we pick a unitary transformation w_1^H that lower-triangularizes $q_1^H D_1$, the $(1,1)$ diagonal block of the new coefficient matrix.

$$(q_1^H D_1)\, w_1^H = \begin{array}{c} \\ m_1 - k_1 \\ k_1 \end{array} \begin{array}{cc} m_1 - k_1 & k_1 \\ \multicolumn{2}{c}{\begin{pmatrix} D_{11} & 0 \\ D_{21} & D_{22} \end{pmatrix}} \end{array}.$$

To multiply the first m_1 columns of the coefficient matrix with w_1^H from the right, we also need to replace the unknowns x_1 by the new (transformed) unknowns

$$w_1 x_1 = \begin{array}{c} m_1 - k_1 \\ k_1 \end{array} \begin{pmatrix} z_1 \\ \hat{x}_1 \end{pmatrix},$$

and do the multiplication $Q_1^H w_1^H$. In summary, equation (5) becomes:

$$\begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} A \begin{pmatrix} w_1^H & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} w_1 & 0 \\ 0 & I \end{pmatrix} x = \begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} b - \begin{pmatrix} 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \end{pmatrix} \tau,$$

which can be simplified to read as follows:

$$\begin{pmatrix} D_{11} & 0 & 0 & 0 & 0 \\ D_{21} & D_{22} & \hat{U}_1V_2^H & \hat{U}_1W_2V_3^H & \hat{U}_1W_2W_3V_4^H \\ P_2Q_{11}^H & P_2\hat{Q}_1^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ P_3R_2Q_{11}^H & P_3R_2\hat{Q}_1^H & P_3Q_2^H & D_3 & U_3V_4^H \\ P_4R_3R_2Q_{11}^H & P_4R_3R_2\hat{Q}_1^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{pmatrix} \begin{pmatrix} z_1 \\ \hat{x}_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \beta_1 \\ \gamma_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \end{pmatrix} \tau,$$

where we have used the partition

$$q_1^H b_1 = \begin{array}{c} m_1 - k_1 \\ k_1 \end{array} \begin{pmatrix} \beta_1 \\ \gamma_1 \end{pmatrix} \quad \text{and} \quad w_1 Q_1 = \begin{array}{c} m_1 - k_1 \\ k_1 \end{array} \begin{pmatrix} Q_{11} \\ \hat{Q}_1 \end{pmatrix}.$$

Now we solve for z_1 from the system of equations $D_{11}z_1 = \beta_1$. We also subtract D_{21} times z_1 from the right-hand side to obtain $\hat{b}_1 = \gamma_1 - D_{21}z_1$.

The remaining z_1 term on the left hand of the new equations has a form that is similar to the τ term on the right hand side and can therefore be merged into a new right hand side term as

$$\begin{pmatrix} 0 \\ 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \end{pmatrix} \hat{\tau}, \quad \text{where} \quad \hat{\tau} = \tau + Q_{11}^H z_1.$$

Now we can discard the first $m_1 - k_1$ rows and columns of the equations to arrive at:

$$\begin{pmatrix} D_{22} & \hat{U}_1V_2^H & \hat{U}_1W_2V_3^H & \hat{U}_1W_2W_3V_4^H \\ P_2\hat{Q}_1^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ P_3R_2\hat{Q}_1^H & P_3Q_2^H & D_3 & U_3V_4^H \\ P_4R_3R_2\hat{Q}_1^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{pmatrix} \begin{pmatrix} \hat{x}_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \hat{b}_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} - \begin{pmatrix} 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \end{pmatrix} \hat{\tau}.$$

Observe that this new system is structurally identical to the one we started with, except that it has a smaller dimension. So let us assume that by the magic of recursion we can solve this system quickly for the unknowns $\hat{x}_1$, x_2, x_3 and x_4. Then we can recover x_1 by computing

$$x_1 = w_1^H \begin{pmatrix} z_1 \\ \hat{x}_1 \end{pmatrix}.$$

This completes the first possibility in the recursive algorithm.

5.1.2 Case of $k_1 \geq m_1$: Merge

The second possibility is that k_1, the number of columns of U_1 is at least as large as m_1 the number of rows (and columns) of D_1. In this case we cannot necessarily induce any zeros in the first few rows by a small transformation from the left. Hence we proceed by merging the first two block rows and columns of the matrix. To do that observe that we can rewrite the system of equations as follows:

$$\begin{pmatrix} \begin{pmatrix} D_1 & U_1V_2^H \\ P_2Q_1^H & D_2 \end{pmatrix} & \begin{pmatrix} U_1W_2 \\ U_2 \end{pmatrix} V_3^H & \begin{pmatrix} U_1W_2 \\ U_2 \end{pmatrix} W_3V_4^H \\ P_3 \begin{pmatrix} Q_1R_2^H \\ Q_2 \end{pmatrix}^H & D_3 & U_3V_4^H \\ P_4R_3 \begin{pmatrix} Q_1R_2^H \\ Q_2 \end{pmatrix}^H & P_4Q_3^H & D_4 \end{pmatrix} \begin{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} b_1 \\ b_2 - P_2\tau \end{pmatrix} \\ b_3 \\ b_4 \end{pmatrix} - \begin{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ P_3 \\ P_4R_3 \end{pmatrix} (R_2\tau).$$

Hence if we make the following definitions (computations)

$$\begin{aligned} \hat{D}_1 &= \begin{pmatrix} D_1 & U_1V_2^H \\ P_2Q_1^H & D_2 \end{pmatrix}, \quad \hat{U}_1 = \begin{pmatrix} U_1W_2 \\ U_2 \end{pmatrix}, \quad \hat{Q}_1 = \begin{pmatrix} Q_1R_2^H \\ Q_2 \end{pmatrix}, \\ \hat{x}_1 &= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \hat{b}_1 = \begin{pmatrix} b_1 \\ b_2 - P_2\tau \end{pmatrix}, \quad \hat{\tau} = R_2\tau, \end{aligned}$$

then we can rewrite the system of equations as follows:

$$\begin{pmatrix} \hat{D}_1 & \hat{U}_1V_3^H & \hat{U}_1W_3V_4^H \\ P_3\hat{Q}_1^H & D_3 & U_3V_4^H \\ P_4R_3\hat{Q}_1^H & P_4Q_3^H & D_4 \end{pmatrix} \begin{pmatrix} \hat{x}_1 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \hat{b}_1 \\ b_3 \\ b_4 \end{pmatrix} - \begin{pmatrix} 0 \\ P_3 \\ P_4R_3 \end{pmatrix} (\hat{\tau}),$$

which is structurally identical to (5). Hence we can proceed to solve it recursively. In case $n = 1$, we have the system $D_1x_1 = b_1 - 0\tau$, which is solved by standard means.

5.2 The Full Algorithm

We now present the full algorithm. We assume that the sequentially semi-separable matrix A is represented by the seven sequences $\{U_i\}_{i=1}^{n-1}$, $\{V_i\}_{i=2}^{n}$, $\{W_i\}_{i=2}^{n-1}$, $\{P_i\}_{i=2}^{n}$, $\{Q_i\}_{i=1}^{n-1}$, $\{R_i\}_{i=2}^{n-1}$ and $\{D_i\}_{i=1}^{n}$ as in (1). We also partition $x = (\, x_i \,)$ and $b = (\, b_j \,)$ such that x_i and b_i have m_i rows. As in the 4×4 example, there are two cases at each step of the recursion.

5.2.1 Case of $n > 1$ and $k_1 < m_1$: Elimination

As in Section 5.1.1, our goal is to do orthogonal eliminations on both sides of A to create an $(m_1 - k_1) \times (m_1 - k_1)$ lower triangular submatrix at the top left corner of A.

We perform orthogonal eliminations by computing QL and LQ factorizations

$$U_1 = q_1 \begin{array}{c} \\ \begin{pmatrix} 0 \\ \hat{U}_1 \end{pmatrix} \end{array} \begin{array}{l} m_1 - k_1 \\ k_1 \end{array} \quad \text{and} \quad \left(q_1^H D_1\right) = \begin{array}{c|cc} & m_1 - k_1 & k_1 \\ m_1 - k_1 & D_{11} & 0 \\ k_1 & D_{21} & D_{22} \end{array} w_1,$$

where q_1 and w_1 are unitary matrices. To complete the eliminations, we also need to apply q_1^H to b_1 and w_1 to Q_1 to obtain

$$q_1^H b_1 = \begin{array}{l} m_1 - k_1 \\ k_1 \end{array} \begin{pmatrix} \beta_1 \\ \gamma_1 \end{pmatrix} \quad \text{and} \quad w_1 Q_1 = \begin{array}{l} m_1 - k_1 \\ k_1 \end{array} \begin{pmatrix} Q_{11} \\ \hat{Q}_1 \end{pmatrix}.$$

Equations (1) have now become

$$\begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} A \begin{pmatrix} w_1^H & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} w_1 & 0 \\ 0 & I \end{pmatrix} x = \begin{pmatrix} q_1^H & 0 \\ 0 & I \end{pmatrix} b - \begin{pmatrix} 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \\ \vdots \\ P_nR_{n-1}\cdots R_2 \end{pmatrix} \tau, \tag{6}$$

As was done in Section 5.1.1, we now orthogonally transform the unknowns x_1 and solve the $(m_1 - k_1) \times (m_1 - k_1)$ lower triangular system of equations. Let $\begin{array}{c} m_1 - k_1 \\ k_1 \end{array} \begin{pmatrix} z_1 \\ \hat{x}_1 \end{pmatrix} = w_1 x_1$. Then the first $m_1 - k_1$ equations of (6) have been simplified to $D_{11} z_1 = \beta_1$. Hence we compute $z_1 = D_1^{-1}\beta_1$ by forward substitution.

We further compute $\hat{b}_1 = \gamma_1 - D_{21} z_1$. This in effect subtracts the D_{21} portion of the columns from the right-hand side. Finally we compute $\hat{\tau} = \tau + Q_{11}^H z_1$. This simple operation merges the previous pending subtraction at the right-hand side and the subtraction of the first $m_1 - k_1$ columns (those corresponding to z_1) from the new right-hand side.

At this stage, we discard the first $m_1 - k_1$ equations and are left with a new linear system of equations

$$\hat{A}\hat{x} = \hat{b} - \begin{pmatrix} 0 \\ P_2 \\ P_3R_2 \\ P_4R_3R_2 \\ \vdots \\ P_nR_{n-1}\cdots R_2 \end{pmatrix} \hat{\tau}$$

with exactly the same form as (1). To see this, we note that among the seven sequences $\{U_i\}_{i=1}^{n-1}$, $\{V_i\}_{i=2}^{n}$, $\{W_i\}_{i=2}^{n-1}$, $\{P_i\}_{i=2}^{n}$, $\{Q_i\}_{i=1}^{n-1}$, $\{R_i\}_{i=2}^{n-1}$ and $\{D_i\}_{i=1}^{n}$, everything remains the same except that U_1, Q_1, and D_1 have been replaced by $\hat{U}_1$, $\hat{Q}_1$, and D_{22}. Among the partitioned unknown subvectors x_i's and right hand side subvectors b_i's, the only changes are that x_1 and b_1 have been replaced by $\hat{x}_1$ and $\hat{b}_1$, respectively. Of course, the new linear system of equations has a strictly smaller dimension, hence we can indeed proceed with this recursion. After we have computed the unknowns x_2 to x_n and the transformed unknowns $\hat{x}_1$, we can recover x_1 using the formula

$$x_1 = w_1^H \begin{pmatrix} z_1 \\ \hat{x}_1 \end{pmatrix}.$$

5.2.2 Case of $k_1 \geq m_1$: Merge

As in Section 5.1.2, we perform merging in this case. In the case $n > 1$ and $m_1 \leq k_1$, we cannot perform eliminations. Instead we merge the first two block rows and columns of A while still maintaining the sequentially semi-separable structure.

We merge the first two blocks by computing

$$\hat{D}_1 = \begin{pmatrix} D_1 & U_1V_2^H \\ P_2Q_1^H & D_2 \end{pmatrix}, \quad \hat{U}_1 = \begin{pmatrix} U_1W_2 \\ U_2 \end{pmatrix}, \quad \text{and} \quad \hat{Q}_1 = \begin{pmatrix} Q_1R_2^H \\ Q_2 \end{pmatrix}.$$

We merge x_1 and x_2 into $\hat{x}_1$, and we merge the right hand sides by computing

$$\hat{b}_1 = \begin{pmatrix} b_1 \\ b_2 - P_2\tau \end{pmatrix} \quad \text{and} \quad \hat{\tau} = R_2\tau.$$

Let $\hat{A}$ and $\hat{b}$ denote the matrix A and the vector b after this merge. We can rewrite (1) equivalently as

$$\hat{A}\hat{x} = \hat{b} - \begin{pmatrix} 0 \\ P_3 \\ P_4R_3 \\ P_5R_4R_3 \\ \vdots \\ P_{n-1}R_{n-2}\cdots R_3 \end{pmatrix} \hat{\tau}.$$

Clearly $\hat{A}$ is again a sequentially semi-separable matrix associated with the seven hatted sequences except that we have reduced the number of blocks from n to $n-1$.

To complete the recursion, we observe as in Section 5.1.2 that if $n = 1$, the equations (1) become the standard linear system of equations and can therefore be solved by standard solution techniques.

5.3 Flop Count

The total flop count for this algorithm can be estimated as follows. For simplicity we assume that compression and merging steps always alternate. We also assume without loss of generality that b has only one column. Then we can show that the leading terms of the flop count are given by

$$2\sum_{i=1}^{n}(m_i + k_{i-1})k_i^2 + (m_i + k_{i-1})^3 + (m_i + k_{i-1})^2 l_{i+1} + k_i^2 m_{i+1} + k_i l_{i+1}(m_{i+1} + l_i + l_{i+2}).$$

When the semi-separable structure is constructed using the construction algorithm presented in Section 6 or its faster variants in Section 10, k_i and l_i will be chosen to be the numerical ranks of the i-th upper-triangular and lower-triangular Hankel blocks, respectively. To get a better feel for the operation count we look at the important case when each block has the same dimension $m_i = m$. Let k and l be the maximum numerical ranks of any upper-triangular and lower-triangular Hankel blocks, respectively. Then above expression is bounded by

$$2N\left(m^2 + m(3k + l) + (3kl + 5k^2) + (2k^3 + k^2l + 2kl^2)/m\right), \tag{7}$$

where $N = n\,m$ is the dimension of the full matrix. We observe that the count is not symmetric in k_i and l_i. Sometimes it is cheaper to compute a URV^T factorization instead if $k > l$. This matter is also covered in [5].

Equation (7) implies that it is possible to chose the optimal block size m to minimize the (upper bound on the) total cost. Indeed, the expression in (7) becomes very large for both very small and very large values of m. Setting the derivative with respect to m in this expression to 0, we see that the total cost is minimized when we choose m to the the integer closest to the unique positive root of the following equation

$$2m + (3k + l) - (2k^3 + k^2l + 2kl^2)/m^2 = 0,$$

or

$$2m^3 + (3k + l)m^2 - (2k^3 + k^2l + 2kl^2) = 0. \tag{8}$$

For $k = l$, this equation simplifies further to

$$2m^3 + 4km^2 - 5k^3 = 0,$$

	size					
$m_i = l_i = k_i$ for all i	256	512	1024	2048	4096	8192
16	0.04	0.08	0.16	0.36	0.67	1.34
32	0.08	0.19	0.42	0.83	1.66	3.44
64	0.18	0.48	1.12	2.36	4.8	9.87
128	0.15	1.01	2.73	6.09	12.91	26.9
Standard Solver (GEPP)	0.15	0.72	4.57	30.46	222.57	3499.46

Table 2: Run-times in seconds for both the fast stable algorithm and standard solver for random sequentially semi-separable matrices with $m_i = k_i = l_i$ for all i.

which has a unique positive root at $m \approx 0.9246k$. With this choice of m, the total cost (7) is about $36Nk^2$ flops. Note that the constant in front of the leading term is not large.

5.4 Experimental run-times

We now report the run-times of this algorithm on a PowerBook G4 running at 400 MHz with 768 MB of RAM. We used the ATLAS BLAS version 3.3.14 and LAPACK version 3 libraries. For comparison we also report the run-times of the standard dense solvers from LAPACK and ATLAS BLAS. All timings are reported in Table 2. The columns are indexed by the actual size ($\sum_{i=1}^{n} m_i$) of the matrix, which range from 256 to 8192. The horizontal rows are indexed by the value of m_i which is set equal to k_i and l_i for all i and ranges from 16 to 128. These are representative for many classes of problems (see section 12). In the last row we report the run-times in seconds of a standard dense (Gaussian elimination) solver from the LAPACK version 3 library running on top of the ATLAS BLAS version 3.3.14. These are highly-tuned routines which essentially run at peak flop rates.

From the table we can see the expected linear dependence on the size of the matrix. The non-quadratic dependence on m_i (and k_i and l_i) seems to be due to the dominance of the low-order complexity terms. For example we observe a *decrease* in run-time when we increase m_i from 64 to 128 for a matrix of size 256! This is because at this size and rank the matrix has no structure and essentially a dense solver (without any of the overhead associated with a fast solver) is being used. There is also a non-linear increase in the run-time when we increase the size from 256 to 512 for $m_i = k_i = l_i = 128$. This is due to the lower over-heads associated with standard solver.

Restricting our attention to the last two rows in Table 2 where $m_i = k_i = l_i = 128$ for all i, we observe that the fast algorithm breaks even with the dense solver for matrices of size between 512 and 1024. (The estimated flop count actually predicts a break-even around matrices of size 940.) For matrices of size 4096 we have speed-ups in excess of 17.2401. Since the standard solver becomes unusually slower for matrices of size 8192 (possibly due to a shortage of RAM) we get a speed-up of 130 at this size. The speed-ups are even better for smaller values of m_i's.

We could further speed up the fast algorithm by using Gaussian elimination with partial pivoting instead of orthogonal transforms. This approach would still be completely stable as long as the dimensions of the diagonal blocks remain small.

5.5 Stability

The algorithm we have presented is backward stable provided the SSS representation is **stable**. By this we mean that $\|W_i\|_2 \leq 1$ and $\|R_i\|_2 \leq 1$. The reasoning is similar to the one presented in [5, 14]. The algorithm implicitly computes an ULV^H factorization, where U and V are unitary and L is lower-triangular. Both U and V are represented as products of elementary unitary transforms. It follows from a straight-forward error analysis that the algorithm is backward-stable provided the SSS representation is stable. Experimental calculations of the backward error has shown that in all instances it is a small multiple of the machine precision, as would be expected for a backward-stable method. The SSS construction algorithm presented in section 6 computes stable SSS representations, as does the fast model reduction algorithm in section 10.6.

However, even if the SSS representation satisfies the much weaker condition $\|W_iW_{i+1}\cdots W_j\|_2 \leq p(n)$ and $\|R_iR_{i-1}\cdots R_j\|_2 \leq p(n)$, where p is a low-degree polynomial in n, then the algorithm is backward-stable to first order in the machine precision. Such **weakly stable** SSS representations are produced by techniques like the fast multipole method. However, they can be made stable by using the fast model reduction algorithm of section 10.6.

6 Constructing sequentially semi-separable matrices

In this section we consider the problem of computing the sequentially semi-separable structure of a matrix given the sequence $\{m_i\}_{i=1}^n$ and a low-rank representation of some off-diagonal blocks. The method presented can be applied to any unstructured matrix, thus proving that any matrix has a sequentially semi-separable structure (of course, k_i and l_i will usually be large in this case, precluding any speed-ups).

6.1 General Construction Algorithm

Let A represent the matrix for which we wish to construct a sequentially semi-separable representation corresponding to the sequence $\{m_i\}_{i=1}^n$, where $\sum m_i = N$, the order of the matrix. Our procedure is similar to that of Dewilde and van der Veen [14]. Since the upper triangular part and lower triangular parts are so similar, we will only describe how to construct the sequentially semi-separable representation of the strictly block upper triangular part of A. The basic idea is to recursively compress off-diagonal blocks into low-rank representations.

Let H_i denote the off-diagonal block

$$H_i = \begin{pmatrix} U_1W_2\cdots W_iV_{i+1}^H & \cdots & U_1W_2\cdots W_{n-1}V_n^H \\ \vdots & \vdots & \vdots \\ U_iV_{i+1}^H & \cdots & U_iW_{i+1}\cdots W_{n-1}V_n^H \end{pmatrix}. \tag{9}$$

Following Dewilde and van der Veen [14] we will call H_i the ith Hankel block. Each H_i is a $\mu_i \times \nu_i$ matrix with $\mu_i = m_1 + \cdots + m_i$ and $\nu_i = m_{i+1} + \cdots + m_n$. Observe that we can obtain H_{i+1} from H_i by dropping the first m_{i+1} columns of H_i and then appending to the resulting matrix the last m_{i+1} rows of H_{i+1}.

To start the construction, we let $H_1 \approx E_1\Sigma_1F_1^H$ denote a low-rank (also called economy) SVD of H_1. That is, we assume that the matrix of singular values Σ_1, is a square invertible matrix, all of whose singular values below a certain threshold have been set to zero. Therefore, E_1 and F_1 have an orthonormal set of columns, but they may not be unitary. Partition F_1 as

$$F_1 = \begin{array}{c} m_2 \\ \nu_2 \end{array} \begin{pmatrix} F_{1,1} \\ F_{1,2} \end{pmatrix},$$

and we pick $U_1 = E_1$ and $V_2 = F_{1,1}\Sigma_1$. Define $\hat{H}_1 = \Sigma_1 F_{1,2}^H$. $\hat{H}_1$ is the portion of H_1 whose SSS construction has not yet been finished. We note that both the column dimension of U_1 and the row dimension of $\hat{H}_1$ are exactly the numerical rank of H_1.

To continue, let Z_2 be the last m_2 rows of H_2 so that

$$H_2 = \begin{pmatrix} U_1 \hat{H}_1 \\ Z_2 \end{pmatrix}.$$

We compute the following SVD

$$\hat{H}_2 = \begin{pmatrix} \hat{H}_1 \\ Z_2 \end{pmatrix} \approx E_2 \Sigma_2 F_2^H.$$

Note that $\hat{H}_2$ is the matrix obtained from H_2 by taking out the matrix U_1 which has already been computed. As before, any singular value of $\hat{H}_2$ below a certain threshold has been set to zero. Partition

$$E_2 = \begin{array}{c} k_1 \\ m_2 \end{array} \begin{pmatrix} E_{2,1} \\ E_{2,2} \end{pmatrix} \quad \text{and} \quad F_2 = \begin{array}{c} m_3 \\ \nu_3 \end{array} \begin{pmatrix} F_{2,1} \\ F_{2,2} \end{pmatrix}.$$

We choose $U_2 = E_{2,2}$, $V_3 = F_{2,1}\Sigma_2$, $W_2 = E_{2,1}$ and set $\hat{H}_2 = \Sigma_2 F_{2,2}^H$. $\hat{H}_2$ is the part of H_2 which requires further SSS construction. We again note that both the column dimension of U_2 and the row dimension of $\hat{H}_2$ are exactly the numerical rank of H_2.

In general, let $\hat{H}_i$ be the remaining part of H_i, and let Z_{i+1} be the last m_{i+1} rows of H_{i+1}, we construct $\hat{H}_{i+1}$ and compute its SVD as follows

$$\hat{H}_{i+1} = \begin{pmatrix} \hat{H}_i \\ Z_{i+1} \end{pmatrix} \approx E_{i+1}\Sigma_{i+1}F_{i+1}^H \quad \text{where} \quad E_{i+1} = \begin{array}{c} k_i \\ m_{i+1} \end{array} \begin{pmatrix} E_{i+1,1} \\ E_{i+1,2} \end{pmatrix}, \quad F_{i+1} = \begin{array}{c} m_{i+2} \\ \nu_{i+2} \end{array} \begin{pmatrix} F_{i+1,1} \\ F_{i+1,2} \end{pmatrix}.$$

As before, we set $U_{i+1} = E_{i+1,2}, V_{i+2} = F_{i+1,1}\Sigma_{i+1}, W_{i+1} = E_{i+1,1}$ and $\hat{H}_{i+1} = \Sigma_{i+1}F_{i+1,2}^H$. Similar to above, both the column dimension of U_{i+1} and the row dimension of $\hat{H}_{i+1}$ are the numerical rank of H_{i+1}. By substituting these formulas back into the sequentially semi-separable representation beginning with H_1, it is straightforward to show that these formulas indeed construct the sss matrix and the construction is numerically stable.

Finally, we note that the sequentially semi-separable representation for the lower triangular part of A can be computed by applying exactly the same procedure above to A^H. The computational costs are similar as well.

This algorithm takes $O(N^2)$ flops, where the hidden constants depend on m_i, k_i and l_i. It is clear from our construction that k_i and l_i are the numerical ranks of their corresponding Hankel blocks. This fact allows us to easily choose the best partitioning that minimizes the overall cost for solving sequentially semi-separable linear systems of equations (see Section 5.3).

Our construction algorithm also makes it clear that an efficient sequentially semi-separable representation can be constructed for any matrix which has low numerical rank on every Hankel block. Hence computational electromagnetics problems involving integral equations can be rapidly solved using an efficient sequentially semi-separable representation.

The algorithm can be implemented to require only $O(N)$ memory locations. This is particularly important in those applications where a large dense structured matrix can be generated (or read from a file) on the fly.

We can replace the use of singular value decompositions with rank-revealing QR factorizations quite easily. This may result in some speed ups with little loss of compression (see [10, 24] and the references therein).

A totally different alternative is to use the recursively semi-separable (RSS) representation presented in the paper by Chandrasekaran and Gu [5]. This is usually easier to compute efficiently, but may be less flexible.

In many important applications the sequentially semi-separable representation needs to be computed only once for a fixed problem size and stored in a file. In such cases the cost of the exact algorithm is not important. Such cases include computing the sequentially semi-separable structure of spectral discretization methods of Greengard and Rokhlin [31, 37] for two-point boundary value problems and that of Kress [12] for integral equations of classical potential theory in two dimensions.

Finally, we note that the sequentially semi-separable representation for the lower triangular part of A can be computed by applying exactly the same procedure above to A^H. The computational costs are similar as well.

This algorithm takes $O(N^2)$ flops, where the hidden constants depend on m_i, k_i and l_i. The algorithm can be implemented to require only $O(N)$ memory locations. This is particularly important in those applications where a large dense structured matrix can be generated (or read from a file) on the fly. Many computational electromagnetics problems involving integral equations fall in this class.

If one is prepared to live dangerously, we can compute U_i, W_i and V_i in time that is much less than $O(N^2)$. For example we see that we can pretend that U_i and V_{i+1} can be computed from the lower-left block of H_i—the one that reads U_iV_{i+1}. Of course, the danger is obvious: we may fail to find all of U_i and V_{i+1}. But for many classes of matrices, especially those that are "heavy" on the diagonal, the chances are good that we will not fail. The necessary condition is that U_i and V_{i+1} be full-rank well-conditioned matrices. Similarly W_{i+1} can be computed from the block that reads $U_iW_{i+1}V_{i+2}^H$. Again, this will only work if U_i and V_{i+1} are full-rank well-conditioned matrices. Note that in this scheme only the banded portion of the matrix needs to be read. To check our guesses we can use nearby blocks. For example we can use the block that reads $U_iW_{i+1}W_{i+2}V_{i+3}^H$ to check our calculations, and even improve them. Such an algorithm, assuming it succeeds, will run in $O(N)$ flops.

7 One pass solution for the Moore-Penrose inverse of a general system

Let be given a general matrix A as before, the entries of A may be blocks, and a vector b conformal to the rows of A. We wish to find a vector x of smallest possible magnitude $\|x\|_2 = \sqrt{\sum \|x_i\|^2}$, which minimizes $\|Ax - b\|_2$. Our goal is to find a backward stable algorithm in the same taste as before, that is using a one-pass, top down algorithm. As before we assume that we dispose of a realization consisting of block matrices $\{U_i\}$, $\{V_i\}$ etc., the only restriction being that these realization matrices must be appropriately conformal to fit the products in the representation. In particular, the diagonal entries $\{D_i\}$ do not have to be square or non-singular. In [13] a general algorithm was given to determine the Moore-Penrose inverse of a general time-varying operator. We specialize the method given in that paper to the matrix case with two

new and important additions:
- the new algorithm is one-pass, top-down;
- explicit use is made of the Chandrasekaran-Gu method.

The one pass character is obtained by making a judiciary use of left and right multiplications by elementary orthogonal matrices. These matrices then combine to semi-separable operators of the same complexity as the original semi-separable data. What makes the Moore-Penrose theory more complex than the mere system reduction is the fact that exact solutions of reduced subsystems may not contribute to the Moore-Penrose solution, except in the special circumstance that all rows in the system are linearly independent. For example, the system of equations

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

has the least squares solution $x = (b_1+b_2)/2$ - the first equation can not be solved independently from the rest of the system. Hence care has to be exercised in converting the system first to a system of row-independent matrices, that is, the co-kernel has to be implicitly determined (see further!).

The strategy that we follow to obtain the reduction to a row-independent form works in two steps, the third step then consists in solving the resulting row-independent system using the methods described earlier (this time actually on a simpler systems). The resulting complexity will roughly be twice that of the original algorithm. Lazy evaluation allows the algorithm to proceed from the upper left corner down to the lower right corner as before. A summary of the procedure is as follows:
1. Using elementary orthogonal (or in case or complex entries unitary) on the columns (on the right of the matrix) we gradually transform the double sided representation into a block upper one, that is one in which the P,Q and R type matrices are all zero. The upper representation will have the same complexity as the original. The result is the representation of a block-upper matrix A_1, related to A through

$$A_1 = Aw$$

where w assembles the column transformations.
2. Using elementary orthogonal transformations on the rows (i.e. on the left) we gradually (lazily) determine co-kernel and range of A_1:

$$A_1 = [u_1 \; u_2] \begin{bmatrix} 0 \\ A_o \end{bmatrix}$$

in which A_o has linearly independent rows. The columns of u_2 then form an orthonormal basis for the range of A_1 and the columns of u_1 for its co-kernel (i.e. the kernel of A_1^H).
3. The combination of the two previous steps has produced

$$A = [u_1 \; u_2] \begin{bmatrix} 0 \\ A_o \end{bmatrix} w^H.$$

With $y = w^H x$ and $\begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} u_1^H b \\ u_2^H b \end{bmatrix}$ we find that the Moore-Penrose solution decomposes in

$$A_o y = \beta_2$$

which is now a system with linearly independent rows that can be solved exactly with the techniques of the preceding sections, leaving an irreducible residue β_1 - the least squares error sought will be $\|\beta_1\|$.

Before detailing the general method we give the algorithm stepwise on a 3×3 example.

For ease of exposition we go through the three 'w' steps first, assemble a complete 'upper' matrix and then proceed through the three steps that work on the rows of the resulting matrix. We use the following notations:

$\mathbf{T}_i$, in which T is an operator (represented by a block matrix) collects the i'th components of a unilateral representation of T in a single block 2×2 matrix

$$\mathbf{T}_i = \begin{bmatrix} A_i & C_i \\ B_i & D_i \end{bmatrix}$$

in which some of the block rows or columns may disappear - dimensions of the blocks have to be congruent of course - the entry $T_{i,j}$ for $i < j$ being given by $T_{i,j} = B_i A_{i+1} \cdots A_{j-1} C_j$ and $T_{i,i} = D_i$;
Splitting of a block, say A, in columns is indicated with a dot notation for a running index, e.g. $A_{\cdot,1}$, $A_{\cdot,2}$ would be a splitting of columns in two blocks, the dimensions being usually clear from the context.
Finally, we use the terms 'realization' and 'representation' indiscriminately to indicate the matrices used in diverse semi-separable representations. In the next sections we assume that all the given representations are of minimal dimensions (they will be if the construction procedures of the previous sections have been used).
Comments are put after a semicolon.

The 'w' procedure

Step 1
We find in sequence:
$M_1 = \cdot$, $r_1 = \cdot$; the start of the procedure is 'empty'
$\mathbf{w}_1 = \begin{bmatrix} \cdot & \cdot \\ Q_1 & d_{w1} \end{bmatrix}$ where d_{w1} is an orthonormal basis for the complement of the column span of Q_1
$[\mathbf{A}_1]_1 = \left[\begin{array}{cc|c} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \hline U_1 & Q_1 & D_1 d_{w1} \end{array}\right]$; this produces the first realization matrices for A_1, the net effect is to project the columns of D_1 to the orthogonal complement of Q_1.
Step 2
$M_2 = Q_1^H Q_1 = r_2^H r_2$ where r_2 is a square matrix making $Q_1 r_2^{-1}$ isometric, if the realization minimal, M_2 will be non-singular.
$\mathbf{w}_2 = \begin{bmatrix} R_2^H & r_2^{-1} b_{w2} \\ Q_2 & d_{w2} \end{bmatrix}$ where $\begin{bmatrix} b_{w2} \\ d_{w2} \end{bmatrix}$ is an orthonormal basis for the complement of the column space of $\begin{bmatrix} r_2 R_2^H \\ Q_2 \end{bmatrix}$; this yields now
$[\mathbf{A}_1]_2 = \left[\begin{array}{cc|c} W_2 & V_2^H & V_2^H d_{w2} \\ & R_2^H & r_2^{-1} b_{w2} \\ \hline U_2 & Q_2 + P_2 M_2 R_2^H & D_2 d_{w2} + P_2 r_2^H b_{w2} \end{array}\right]$
and allows us to construct the first two block columns of A_1:

$$A_1 = \left[\begin{array}{ccc} D_1 d_{w1} & [U_1\ Q_1] \begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} b_{w2} \end{bmatrix} & ? \\ 0 & D_2 d_{w2} + P_2 r_2^H b_{w2} & ? \\ \hline 0 & 0 & ? \end{array}\right]$$

Step 3
$M_3 = Q_2^H Q_2 + R_2 M_2 R_2^H = r_3^H r_3$, in which r_3 is square non-singular, since we assume minimal realizations and hence M_3 is non-singular.
$\mathbf{w}_3 = \begin{bmatrix} R_3^H & r_3^{-1} b_{w3} \\ Q_3 & d_{w3} \end{bmatrix}$ where $\begin{bmatrix} b_{w3} \\ d_{w3} \end{bmatrix}$ forms an orthonormal basis for the complement of the column space of $\begin{bmatrix} r_3 R_3^H \\ Q_3 \end{bmatrix}$, yielding the third set of realization blocks for A_3:

$$[\mathbf{A}_1]_3 = \left[\begin{array}{cc|c} W_3 & V_3^H Q_3 & V_3^H d_{w3} \\ & R_3^H & r_3^{-1} b_{w3} \\ \hline U_3 & Q_3 + P_3 M_3 R_3^H & D_3 d_{w3} + P_3 r_3^H b_{w3} \end{array}\right]$$

and finally the full upper conversion:

$$A_1 = \left[\begin{array}{ccc} D_1 d_{w1} & [U_1\ Q_1]\begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} b_{w2} \end{bmatrix} & [U_1\ Q_1]\begin{bmatrix} W_2 & U_2^H Q_2 \\ 0 & R_2^H \end{bmatrix}\begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} d_{w3} \end{bmatrix} \\ 0 & D_2 d_{w2} + P_2 r_2^H b_{w2} & [U_2\ Q_2 + P_2 M_2 R_2^H]\begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix} \\ \hline 0 & 0 & D_3 d_{w3} + P_3 r_3^H b_{w3} \end{array}\right]$$

This terminates the w transformation for the 3×3 case. It should be clear from the general formulas that the procedure goes on in the same recursive way for larger matrices, there is no growth in dimension of the entries, they are all roughly twice the size of the originals, and still minimal.

The conversion to linearly independent rows

We use a different ordering of rows and columns than in the theory, the symbols and their indexing should be self-explaining.
Step 1: QR decomposition on $[D_1 d_{w1} | U_1 Q_1]$:
$\begin{bmatrix} \delta_{11}^H \\ \delta_{21}^H \\ \beta_1^H \end{bmatrix} [D_1 d_{w1}\,|\,U_1 Q_1] = \begin{bmatrix} 0 & 0 \\ d_{o1} & b_{o1} \\ 0 & y_2 \end{bmatrix}$ where $Q_1^H = \begin{bmatrix} \delta_{11}^H \\ \delta_{21}^H \\ \beta_1^H \end{bmatrix}$ is the 'Q' of the QR decomposition (for later convenience and consistency we've put the bottom row of zeros in the R factor on top), d_{o1} and y_2 have linearly independent rows by construction (they can be in echelon form), and the result of step 1 applied to the whole matrix by embedding the elementary transformation Q_1^H is

$$\left[\begin{array}{ccc} 0 & 0 & 0 \\ d_{o1} & b_{o1}\begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} d_{w2} \end{bmatrix} & b_{o1}\begin{bmatrix} W_2 & U_2^H Q_2 \\ 0 & R_2^H \end{bmatrix}\begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w_3} \end{bmatrix} \\ \hline 0 & y_2\begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} b_{w2} \end{bmatrix} & y_2\begin{bmatrix} W_2 & U_2^H Q_2 \\ 0 & R_2^H \end{bmatrix}\begin{bmatrix} V_3^H d_{w3} r_3^{-1} b_{w3} \end{bmatrix} \\ \hline\hline 0 & D_2 d_{w2} + P_2 r_2^H b_{w2} & [U_2\ Q_2 + P_2 M_2 R_2^H]\begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix} \\ \hline 0 & 0 & D_3 d_{w3} + P_3 r_3^H b_{w3} \end{array}\right]$$

and the block row containing y_2 goes to the next stage while the first two block rows will remain untouched in the next steps.
Step 2
Let now, for ease of notation and consistent with the previous theory, $c_2 = \begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} b_{w2} \end{bmatrix}$ and $a_2 = \begin{bmatrix} W_2 & U_2^H Q_2 \\ 0 & R_2^H \end{bmatrix}$, the step consists again in finding a QR factorization:

$$Q_2^H \begin{bmatrix} y_2 c_2 & y_2 a_2 \\ d_2 & b_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ d_{o2} & b_{o2} \\ 0 & y_3 \end{bmatrix};$$

in which Q_2^H has the block decomposition:

$$Q_2^H = \begin{bmatrix} \delta_{21} & \gamma_{21}^H \\ \delta_{22}^H & \gamma_{22}^H \\ \beta_2^H & \alpha_2^H \end{bmatrix}$$

producing as result for the second step:

$$\left[\begin{array}{ccc} 0 & 0 & 0 \\ d_{o1} & b_{o1}\begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} b_{w2} \end{bmatrix} & b_{o1}\begin{bmatrix} W_2 & U_2^H Q_2 \\ & R_2^H \end{bmatrix}\begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix} \\ \hline 0 & 0 & 0 \\ 0 & d_{o2} & b_{o2}\begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix} \\ \hline 0 & 0 & y_3 \begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix} \\ 0 & 0 & D_3 d_{w3} + P_3 r_3^H b_{w3} \end{array}\right].$$

The final step will reduce the two last block rows to row-independent:

$$Q_3^H \begin{bmatrix} y_3 \begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix} \\ D_3 d_{w3} + P_3 r_3^H b_{w3} \end{bmatrix} = \begin{bmatrix} 0 \\ d_{o3} \end{bmatrix},$$

in which the last block consists of linearly independent rows to yield the final result, now written in shorthand:

$$\left[\begin{array}{ccc} 0 & 0 & 0 \\ d_{o1} & b_{o1}c_2 & b_{o1}a_2c_3 \\ \hline 0 & 0 & 0 \\ 0 & d_{o2} & b_{o2}c_3 \\ \hline 0 & 0 & 0 \\ 0 & 0 & d_{o3} \end{array}\right] \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} ?=? \left[\begin{array}{c} \beta_{10} \\ \beta_{11} \\ \hline \beta_{20} \\ \beta_{21} \\ \hline \beta_{30} \\ \beta_{33} \end{array}\right]$$

in which

$$c_2 = \begin{bmatrix} V_2^H d_{w2} \\ r_2^{-1} b_{w2} \end{bmatrix},\ a_2 = \begin{bmatrix} W_2 & U_2^H Q_2 \\ & R_2^H \end{bmatrix},\ c_3 = \begin{bmatrix} V_3^H d_{w3} \\ r_3^{-1} b_{w3} \end{bmatrix},\ d_{o3} = D_3 d_{w3} + P_3 r_3^H b_{w3},$$

are realization matrices for the resulting A_o and the right members have been duly modified as dictated by the transformation. Hence, the quantities β_{10}, β_{20} remain to produce the Moore-Penrose square error:

$$\epsilon_{MPI}^2 = \|\beta_{10}\|^2 + \|\beta_{20}\|^2 + \|\beta_{30}\|^2,$$

while y_1, y_2 and y_3 will now be uniquely determined by solving a unilateral semi-separable system given by

$$\begin{bmatrix} d_{o1} & b_{o1}c_2 & b_{o1}a_2c_3 \\ & d_{o2} & b_{02}c_3 \\ & & d_{o3} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \beta_{11} \\ \beta_{21} \\ \beta_{31} \end{bmatrix}$$

and this system can now be solved by the direct method explained earlier in this paper (see the motivation further on).

It should now be clear that the two steps of the algorithm just described can be done in a top-down, left-right fashion with 'lazy' execution, and only the state space description has to be handled, of course. In fact, one can say alternatively, that the state space descriptions are treated in increasing index order. This will also be true for the algorithm described next.

Proofs and connection with previous theory

We use the general block-matrix representation of [13]. In contrast to the general time-varying system notation we always take the '11' block as the origin (in the time-varying systems notation it is the '00' block for obvious reasons). In cases of doubt, the '11' block is singled out by putting a square around it. Matrices are block-upper if their block entries A_{ij} with $i > j$ are zero. Blocks may be empty, empty blocks have either a zero number of rows, a zero number of columns or both. We represent them by '$-$', '$|$' or '$\cdot$' respectively. By convention, the product of an empty matrix of dimensions $m \times 0$ with an empty matrix of dimensions $0 \times n$ is an $m \times n$ matrix of zeros. (The scalar 0 has dimension 1×1 of course). A special matrix is the shift matrix Z with the general (block-) form

$$\begin{bmatrix} \boxed{|} & I & & & \\ & 0 & I & & \\ & & \ddots & \ddots & \\ & & & 0 & I \\ & & & & - \end{bmatrix}$$

Notice the empty entries on the main diagonal, due to the index shift. The dimensions of the blocks in this matrix may vary from entry to entry, but the blocks indicated by I are all square. The shift matrix is nothing but a unit matrix - the underlying 'skeleton' is a block unit matrix. Applied to the left it shifts a vector 'upwards' by one index, applied to the right, it will shift it to the right (in the sense of the natural vector ordering, it is 'anticausal' when applied to the left and 'causal' when applied to the right of a vector). If we now define diagonal matrices P, Q, R etc... (with empty entries for the missing blocks), e.g. $P = \text{diag}[\boxed{\cdot}, P_2, \cdots, P_n]$, then we find a straight representation for T as

$$A = PZ^H(I - RZ^H)^{-1}Q^H + D + UZ(I - WZ)^{-1}V^H.$$

A decomposes into a 'strictly lower' matrix A_ℓ given by $PZ^H(I - RZ^H)^{-1}Q^H$ and an upper matrix given by $D + UZ(I - WZ)^{-1}V^H$. Our strategy to determine the nullspace of A with a computational complexity equal to the original state space formalism will now consist in two steps:
(1) first we compute a unitary matrix w and A_1 such that $A_1 = Aw$ is block upper,
(2) next we look for an isometric matrix u_1 such that $u_1 A_1 = 0$ (by isometric we mean that $u_1^H u_1 = I$.

Although step one may seem superfluous at first, it seems necessary because it converts the representation of A into the block upper representation needed for the second step. Every entry in A can make it singular - a determination of the co-kernel of T will necessarily involve all its entries. In step two we then compute a unitary matrix $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ such that

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} A = \begin{bmatrix} 0 \\ A_o \end{bmatrix}$$

and A_o has linearly independent rows (i.e. A_o has a right inverse, or $\ker(\cdot A) = 0$ - A_o has zero co-kernel). We shall see that both steps can be done in purely top-down fashion. The final procedure will then be to solve the system of linearly independent rows.

Step I: making the matrix block upper

In this step we compute w such that $A_1 = Aw$ is block upper. This step is nothing but a straightforward conversion of the double sided representation for A (in function of two shifts Z and Z^H) into an equally efficient representation in a single shift of 'Z' type. The purpose of w is simply to cancel out the strictly lower part of the original matrix A - the part represented by $A_\ell = PZ^H(I - RZ^H)^{-1}Q^H$. We shall soon see, by direct computation, that w has a representation of the form

$$w = d_w + QZ(I - R^H Z)^{-1} b'_w$$

i.e. w^H shares R and Q^H with A_ℓ, d_w and b'_w have to be computed (the prime anticipates on the computation further) . (The principle of sharing parts of the representation is fairly universally applicable in cancellation or factorization theory). Specializing to the k'th block and realizing that w should possess a unitary realization since it is a unitary matrix, we look for matrices r_k, b_{wk} and d_{wk} such that

$$\begin{bmatrix} r_k R_k^H r_{k+1}^{-1} & b_{wk} \\ Q_k r_{k+1}^{-1} & d_{wk} \end{bmatrix}$$

is an orthonormal matrix, for each relevant k.
An alternative, equivalent but somewhat simpler realization for w is given by

$$\begin{bmatrix} R_k^H & r_k^{-1} b_{wk} \\ Q_k & d_{wk} \end{bmatrix}$$

so that we have

$$w = d_w + QZ(I - R^H Z)^{-1} r^{-1} b_w.$$

(hence $b'_w = r^{-1} b_w$).

Introducing w in the product Aw we find

$$\begin{aligned} Aw &= [PZ^H(I - RZ^H)^{-1}Q^H + D + UZ(I - WZ)^{-1}V^H] \\ &\quad \cdot [d_w + QZ(I - R^H Z)^{-1} r^{-1} b_w], \end{aligned}$$

in which one 'mixed' quadratic term occurs, which we split:

$$\begin{aligned} &PZ^H(I - RZ^H)^{-1} Q^H Q Z (I - R^H Z)^{-1} r^{-1} b_w \\ &= P(I - Z^H R)^{-1} Z^H R r^H b_w + P r^H b_w + PMR^H Z (I - R^H Z)^{-1} r^{-1} b_w \end{aligned}$$

in which $M = r^H r$ has to satisfy the 'algebraic Lyapunov-Stein equation'

$$RMR^H + Q^H Q = ZMZ^H$$

or specialized to the k'th stage

$$R_k M_k R_k^H + Q_k^H Q_k = M_{k+1}.$$

Hence we find

$$\begin{aligned} Aw &= PZ^H(I - RZ^H)^{-1}(Q^H d_w + R r^H b_w) \\ &+ Dd_w + P r^H b_w \\ &+ UZ(I - WZ)^{-1} V^H d_w + (Q + PMR^H)(I - R^H Z)^{-1} r^{-1} b_w \\ &+ UZ(I - WZ)^{-1} V^H Q Z (I - R^H Z)^{-1} r^{-1} b_w \end{aligned}$$

in which the first term is zero because of the orthogonality in the realization of w, and the subsequent terms constitute the desired realization of A_1:

$$A_1 = (Dd_w + P r^H b_w) + [UZ \;\; Q + PMR^H Z] \begin{bmatrix} I - WZ & V^H Q \\ 0 & I - R^H Z \end{bmatrix}^{-1} \begin{bmatrix} V^H d_w \\ r^{-1} b_w \end{bmatrix}.$$

Since the Lyapunov-Stein equation can be computed top-down, this realization can be obtained in a top-down fashion.

Step II: the determination of the range and the co-kernel

In this step we compute a realization for an orthonormal matrix u which is such that

$$A_1 = [u_1 \; u_2] \begin{bmatrix} 0 \\ A_o \end{bmatrix}$$

i.e. u_1^H spans the co-kernel of A_1 and u_2 its range. As in the previous step, the realization for u will turn out to have the same complexity as the realization for A_1 (and A). Let, for shortness, the realization for A_1 be denoted as

$$A_1 = a + bZ(I - aZ)^{-1}c$$

and let the realizations for u and T_o be given by

$$\begin{aligned} u &= D_u + B_u Z(I - A_u Z)^{-1} C_u \\ A_o &= D_o + B_o Z(I - aZ)^{-1} c \end{aligned}$$

where this time we assume that A_o inherits $\{a, c\}$ from A_1, motivated by the computation that follows (or by the generalized Beurling-Lax theory of [13]). The realization for u as an orthonormal operator can always be choosen to form orthonormal matrices by themselves, hence we shall require, for each k, that

$$\begin{bmatrix} A_{uk} & C_{uk} \\ B_{uk} & D_{uk} \end{bmatrix}$$

be orthogonal matrices - they will follow from the procedure. Writing out

$$u^H A_1 = \begin{bmatrix} 0 \\ A_o \end{bmatrix}$$

we find

$$(D_u^H + C_u^H (I - Z^H A_u^H)^{-1} B_u^H)(d + bZ(I - aZ)^{-1}c) = d_o + b_o Z(I - aZ)^{-1} c.$$

Again we split the mixed quadratic term, introducing a new block diagonal matrix y:

$$(I - Z^H A_u^H)^{-1} Z^H B_u^H b Z(I - aZ)^{-1} = Z^H A_u^H (I - Z^H A_u^H)^{-1} y + y(I - aZ)^{-1}$$

in which y satisfies a Lyapunov-Stein type recursion:

$$ZyZ^H = B_u^H b + A_u^H y a$$

or, per stage k

$$y_{k+1} = B_{uk}^H b_k + A_{uk}^H y_k a_k.$$

Equating the two sides now produces the four equations (for each k)

$$\begin{aligned} \begin{bmatrix} 0 \\ d_{ok} \end{bmatrix} &= \begin{bmatrix} D_{u_1 k}^H \\ D_{u_2 k}^H \end{bmatrix} d_k + \begin{bmatrix} C_{u_1 k}^H \\ C_{u_2 k}^H \end{bmatrix} y_k c_k \\ \begin{bmatrix} 0 \\ b_{ok} \end{bmatrix} &= \begin{bmatrix} D_{u_1 k}^H \\ D_{u_2 k}^H \end{bmatrix} b_k + \begin{bmatrix} C_{u_1 k}^H \\ C_{u_2 k}^H \end{bmatrix} y_k a_k \\ 0 &= A_{uk}^H y_k c_k + B_{uk}^H d_k \\ y_{k+1} &= B_{uk}^H b_k + A_{uk}^H y_k a_k \end{aligned}$$

or in matrix form

$$\begin{bmatrix} C_{u_1 k}^H & D_{u_1 k}^H \\ A_{uk}^H & B_{uk}^H \\ C_{u_2 k}^H & D_{u_2 k}^H \end{bmatrix} \begin{bmatrix} y_k a_k & y_k c_k \\ b_k & d_k \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ y_{k+1} & 0 \\ b_{ok} & d_{ok} \end{bmatrix}.$$

In these expressions it must be so that all y_k and all d_{ok} have independent rows, or in kernel terms: $\ker(\cdot y_{k+1}) = 0$ and $\ker(\cdot d_{ok}) = 0$. (the property for y follows from the inner-outer factorization theory of [13], while the property for d_o follows from its outerness). The actual numerical algorithm produces a canonical lower echelon form using unitary transformations on the rows of

$$\begin{bmatrix} y_k a_k & y_k c_k \\ b_k & d_k \end{bmatrix},$$

of the form:

$$\begin{bmatrix} \cdots & & & & & & \\ '0' & '0' & & & & & \\ '*' & * & '0' & 0 & & & \\ '*' & * & '*' & * & '0' & 0 & \\ '*' & * & '*' & * & '*' & * & '0' \end{bmatrix},$$

starting from the rightmost entries of the bottom row, working upwards, and in which ultimately the top rows are eventually zero (they may disappear) - also all the entries marked between $'*'$ can disappear. The resulting orthonormal transformation matrix will then be subdivided as

$$\begin{bmatrix} C_{u_1k}^H & D_{u_1k}^H \\ A_{uk}^H & B_{uk}^H \\ C_{u_2k}^H & D_{u_2k} \end{bmatrix}$$

because its block entries $C_{u_1k}^H\ D_{u_1k}^H$ correspond to the zero rows of the left hand side echelon form.

Working these results back into the original form, we find for the k'th matrix (on which the echelon form has to be computed and with an appropriate decomposition of y))

$$\begin{bmatrix} y_{1k} & y_{2k} & 0 \\ 0 & 0 & I \end{bmatrix} \left[\begin{array}{cc|c} W_k & V_k^H Q_k & V_k^H d_{wk} \\ 0 & R_k^H & r_k^{-1} b_{wk} \\ \hline U_k & Q_k + P_k M_k R_k^H & D_k d_{wk} + P_k r_k^* b_{wk} \end{array}\right]$$

with

$$M_{k+1} = Q_k^H Q_k + R_k M_k R_k^H$$

$M_k = r_k^H r_k$ and $\begin{bmatrix} b_{wk} \\ d_{wk} \end{bmatrix}$ defined by the orthonormal completion of

$$\begin{bmatrix} r_k R_k r_{k+1}^{-1} & \underline{b_{wk}} \\ Q_k r_{k+1}^{-1} & \underline{d_{wk}} \end{bmatrix}.$$

To terminate the proof we still have to show that a system with independent rows can be solved using the methods explained in the earlier chapters. This is achieved by the following properties.

Lemma

Suppose

$$(R_{11} \quad R_{21} \quad R_{22})$$

has linearly independent rows with R_{11} square. Then R_{11} is invertible and the Moore-Penrose inverse for

$$\begin{bmatrix} R_{11} & \\ R_{21} & R_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

is given by $x_1 = R_{11}^{-1} b_1$ plus the M.-P. solution of

$$R_{22} x_2 = b_2 - R_{21} R_{11}^{-1} x_1.$$

Proof

The invertibility of R_{11} is obvious, and as a consequence, R_{22} has linearly independent rows. Let $R_{22} = [R'_{22}\ 0]q$ be an L-Q decomposition of R_{22}, in which R'_{22} is square and invertible. With $Q = \begin{bmatrix} I & \\ & q \end{bmatrix}$, we have

$$\| Rx - b \|_2 = \| RQ^H Qx - b \|_2,$$

and furthermore, with $y = Qx$,

$$\|Rx - b\|_2 = \| \begin{bmatrix} R_{11} & 0 & 0 \\ R_{21} & R'_{22} & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \|_2$$

Moore-Penrose puts $y_3 = 0$ (for minimal norm of the argument) and solves

$$\begin{bmatrix} R_{11} & 0 \\ R_{21} & R'_{22} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

exactly to make the error zero as well. We obtain: $R_{11}y_1 = b_1$ and $R_{21}y_1 + R'_{22}y_2 = b_2$, from which the statement of the lemma follows.
QED

The procedure of the paper now applies, with the additional remark that in each step of the procedure, equations of the type "$D_{11}z_1 = \beta_1$" will have a square and invertible "D_{11}" and hence will be solvable.

8 A more general representation

The sequentially-semiseparable representation can be improved in various ways to better suit the problem at hand. For matrices which are essentially discretizations of functions of the form $\log \|x - y\|$ or $1/\|x - y\|$, it is often better to replace the block diagonal part by a block-tridiagonal part. This is because the bottom left corner of the Hankel-block H_i is typically a full-rank matrix in these applications, and they will end-up being merged into the diagonal blocks anyway in the fast solver. We can save on this unnecessary merge by generalizing the representation to allow for a block tridiagonal part. This will also permit the rank of the off-diagonal blocks to be much lower. For example a block 5×5 matrix would look like this now

$$A = \begin{pmatrix} D_1 & B_1 & U_1V_3^H & U_1W_2V_4^H & U_1W_2W_3V_5^H \\ C_1 & D_2 & B_2 & U_2V_4^H & U_2W_3V_5^H \\ P_3Q_1^H & C_2 & D_3 & B_3 & U_3V_5^H \\ P_4R_2Q_1^H & P_4Q_2^H & C_3 & D_4 & B_4 \\ P_5R_3R_2Q_1^H & P_5R_3Q_2^H & P_5Q_3^H & C_4 & D_5 \end{pmatrix}$$

This is slightly inelegant since the indices are no longer contiguous for the off-diagonal terms. One can go to a more elegant (and general) representation by allowing overlapping diagonal blocks. In essence we would redefine the diagonal blocks as follows

$$D_i \leftarrow (\, C_{i-1} \quad D_i \quad B_i \,).$$

However this leads to the development of a cumbersome non-classical partitioning notation, which we avoid for the sake of simplicity.

The fast algorithms of this paper can be modified to accommodate this more general structure.

9 A fast multi-way algorithm

So far we have presented the solver as proceeding from the top-left to the bottom-right of the matrix. We always try to compress the first block row, and then merge it with the second-block row. A moments thought will convince you that this is a sub-optimal choice. For example, after compressing the first block-row, and before merging with the second block-row, it might be wise to compress the second-block row first. This has the advantage of compressing a block-row with potentially much smaller rank. Of course, one can extend this idea to consider all the block-rows. However the sequentially semi-separable structure is not very efficient for such a scheme. The reason is that the k_i's represent the ranks of the Hankel blocks H_i which is not relevant in such a scheme. An example will clarify the claim.

Consider the sequentially semi-separable matrix

$$A = \begin{pmatrix} D_1 & 0 & \cdots & 0 & B_1 \\ 0 & D_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & D_{n-1} & 0 \\ C_1 & 0 & \cdots & 0 & D_n \end{pmatrix}.$$

In this case $k_i = \text{rank}(B_1)$ and $l_i = \text{rank}(C_1)$ for all i. However, the second block row for example clearly requires no compression step. Now suppose we block re-order the problem and obtain

$$\Pi A \Pi^T = \begin{pmatrix} D_2 & 0 & \cdots & 0 & 0 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & D_{n-1} & 0 & 0 \\ 0 & \cdots & 0 & D_n & C_1 \\ 0 & \cdots & 0 & B_1 & D_1 \end{pmatrix}.$$

Now $k_i = 0$ and $l_i = 0$ for all i except $i = n-1$, for which they take on the ranks of C_1 and B_1 respectively. In short, re-orderings can dramatically change the ranks of the Hankel-blocks.

Now we can seemingly proceed in two ways. Either we can find the best ordering before we start the algorithm, or, we can do it on-line as the algorithm proceeds. The first approach is difficult to implement in practice. The second is not suitable with the sequentially semi-separable representation.

In a future paper we will present a different representation (the hierarchically semi-separable representation) that is more suitable to this approach, and yields very fast algorithms.

10 Fast updating

We now develop some useful facts about sequentially semi-separable matrices that are needed in section 12. In particular we need the ability to rapidly update the sequentially semi-separable representation of a matrix under various conditions. Some of these are discussed here.

10.1 Merging and Splitting SSS blocks

We first consider the simple operations of merging (small) blocks and splitting (large) blocks in the SSS representation. According to the discussions in Section 5.3, an SSS representation is the most efficient when the block sizes are chosen in relation to the numerical ranks of Hankel blocks. Blocks that are too small or too large could result in an inefficient algorithm for the solution of the corresponding linear systems of equations. This issue is especially important when the SSS structure undergoes repeated modifications (see [**?**]).

We first consider block merging. In the following example, we have merged the second and third blocks in a 4×4 SSS matrix into one single block:

$$\begin{aligned} A &= \begin{pmatrix} D_1 & U_1V_2^H & U_1W_2V_3^H & U_1W_2W_3V_4^H \\ P_2Q_1^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ P_3R_2Q_1^H & P_3Q_2^H & D_3 & U_3V_4^H \\ P_4R_3R_2Q_1^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{pmatrix} \\ &= \begin{pmatrix} D_1 & U_1\begin{pmatrix} V_2^H & W_2V_3^H \end{pmatrix} & U_1\left(W_2W_3\right)V_4^H \\ \begin{pmatrix} P_2 \\ P_3R_2 \end{pmatrix} Q_1^H & \begin{pmatrix} D_2 & U_2V_3^H \\ P_3Q_2^H & D_3 \end{pmatrix} & \begin{pmatrix} U_2W_3 \\ U_3 \end{pmatrix} V_4^H \\ P_4\left(R_3R_2\right)Q_1^H & P_4\begin{pmatrix} R_3Q_2^H & Q_3^H \end{pmatrix} & D_4 \end{pmatrix}. \end{aligned}$$

In general, To merge blocks i and $i+1$ in a given SSS representation, we keep all other blocks unchanged and use the following matrices to represent the new i-th block:

$$\begin{aligned} D_i^{\mathbf{new}} &= \begin{pmatrix} D_i & U_iV_{i+1}^H \\ P_{i+1}Q_i^H & D_{i+1} \end{pmatrix}, \ U_i^{\mathbf{new}} = \begin{pmatrix} U_iW_{i+1} \\ U_{i+1} \end{pmatrix}, \ P_i^{\mathbf{new}} = \begin{pmatrix} P_i \\ P_{i+1}R_i \end{pmatrix}, \\ (V_i^{\mathbf{new}})^H &= \begin{pmatrix} V_i^H & W_iV_{i+1}^H \end{pmatrix}, \ W_i^{\mathbf{new}} = W_iW_{i+1}, \ R_i^{\mathbf{new}} = R_{i+1}R_i, \ (Q_i^{\mathbf{new}})^H = \begin{pmatrix} R_{i+1}Q_i^H & Q_{i+1}^H \end{pmatrix}. \end{aligned}$$

Now we consider the issue of splitting one block into two. To split block i into two blocks for a given SSS representation, we can simply keep all other blocks unchanged and use the above merging formulas reversely to get the equations for the new i and $i+1$ blocks:

$$\begin{gathered} \begin{pmatrix} D_i^{\mathbf{new}} & U_i^{\mathbf{new}}\left(V_{i+1}^{\mathbf{new}}\right)^H \\ P_{i+1}^{\mathbf{new}}\left(Q_i^{\mathbf{new}}\right)^H & D_{i+1}^{\mathbf{new}} \end{pmatrix} = D_i, \ \begin{pmatrix} U_i^{\mathbf{new}}W_{i+1}^{\mathbf{new}} \\ U_{i+1}^{\mathbf{new}} \end{pmatrix} = U_i, \ \begin{pmatrix} P_i^{\mathbf{new}} \\ P_{i+1}^{\mathbf{new}}R_i^{\mathbf{new}} \end{pmatrix} = P_i, \\ \left(\left(V_i^{\mathbf{new}}\right)^H \quad W_i^{\mathbf{new}}\left(V_{i+1}^{\mathbf{new}}\right)^H\right) = V_i^H, \ W_i^{\mathbf{new}}W_{i+1}^{\mathbf{new}} = W_i, \ R_{i+1}^{\mathbf{new}}R_i^{\mathbf{new}} = R_i, \ \left(R_{i+1}^{\mathbf{new}}\left(Q_i^{\mathbf{new}}\right)^H \quad \left(Q_{i+1}^{\mathbf{new}}\right)^H\right) = Q_i^H. \end{gathered}$$

To solve these equations, we partition the matrices for the old i-th block conformally with the two new blocks as

$$D_i = \begin{pmatrix} D_i^{11} & D_i^{12} \\ D_i^{21} & D_i^{22} \end{pmatrix}, \ U_i = \begin{pmatrix} U_i^1 \\ U_i^2 \end{pmatrix}, \ P_i = \begin{pmatrix} P_i^1 \\ P_i^2 \end{pmatrix}, \ V_i^H = \left(\left(V_i^1\right)^H \quad \left(V_i^2\right)^H\right), Q_i^H = \left(\left(Q_i^1\right)^H \quad \left(Q_i^2\right)^H\right).$$

These equations allow us to identify

$$D_i^{\mathbf{new}} = D_i^{11}, \ D_{i+1}^{\mathbf{new}} = D_i^{22}, \ U_{i+1}^{\mathbf{new}} = U_i^2, \ P_i^{\mathbf{new}} = P_i^1, \ V_i^{\mathbf{new}} = V_i^1, \ Q_{i+1}^{\mathbf{new}} = Q_i^2.$$

The remaining matrices satisfy

$$\begin{pmatrix} \left(V_i^2\right)^H & W_i \\ D_i^{12} & U_i^1 \end{pmatrix} = \begin{pmatrix} W_i^{\mathbf{new}} \\ U_i^{\mathbf{new}} \end{pmatrix} \left(\left(V_{i+1}^{\mathbf{new}}\right)^H \quad W_{i+1}^{\mathbf{new}} \right), \ \begin{pmatrix} P_i^2 & D_i^{21} \\ R_i & \left(Q_i\right)^H \end{pmatrix} = \begin{pmatrix} P_{i+1}^{\mathbf{new}} \\ R_{i+1}^{\mathbf{new}} \end{pmatrix} \left(R_i^{\mathbf{new}} \quad \left(Q_i^{\mathbf{new}}\right)^H \right).$$

By factorizing the left hand side matrices using numerical tools such as the SVD and rank revealing QR factorizations, these two equations allow us to compute an effective representation of those remaining matrices for the new blocks.

10.2 Adding two sequentially semi-separable matrices

We now establish that if two sequentially semi-separable matrices are added together we get another sequentially semi-separable matrix. Let A and B be two sequentially semi-separable matrices that are commensurately partitioned; that is, $m_i(A) = m_i(B)$. Then, it is easy to check, that the sum $A+B$, is a sequentially semi-separable matrix with representation given by the sequences

$$\begin{aligned}
U_i(A+B) &= \begin{pmatrix} U_i(A) & U_i(B) \end{pmatrix}, \quad V_i(A+B) = \begin{pmatrix} V_i(A) & V_i(B) \end{pmatrix}, \quad D_i(A+B) = D_i(A) + D_i(B), \\
W_i(A+B) &= \begin{pmatrix} W_i(A) & 0 \\ 0 & W_i(B) \end{pmatrix}, \quad R_i(A+B) = \begin{pmatrix} R_i(A) & 0 \\ 0 & R_i(B) \end{pmatrix}, \\
P_i(A+B) &= \begin{pmatrix} P_i(A) & P_i(B) \end{pmatrix}, \quad Q_i(A+B) = \begin{pmatrix} Q_i(A) & Q_i(B) \end{pmatrix}.
\end{aligned}$$

Note that the computed representation of the sum might be inefficient, in the sense that the ranks of the Hankel blocks are assumed to increase additively, whereas, in some cases the rank might be far smaller. Ideally, these formulas should be followed by a rank-reduction (or model reduction) step (see section 10.6).

10.3 Low-rank plus sequentially semi-separable matrices

We next establish that the sum of a sequentially semi-separable matrix A and a rank-k matrix XY^H is another sequentially semi-separable matrix $B = A + XY^H$. This follows from section 10.2 since a rank-k matrix is a sequentially semi-separable matrix with Hankel-rank k. First some notation. Let

$$X = \begin{array}{c} m_1 \\ \vdots \\ m_n \end{array} \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} \quad \text{and} \quad Y = \begin{array}{c} m_1 \\ \vdots \\ m_n \end{array} \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix}.$$

Then we have that

$$\begin{aligned}
U_i(XY^H) &= X_i, \quad V_i(XY^H) = Y_i, \quad D_i(XY^H) = X_i Y_i^H, \quad W_i(XY^H) = I_k, \\
P_i(XY^H) &= X_i, \quad Q_i(XY^H) = Y_i, \quad R_i(XY^H) = I_k.
\end{aligned}$$

10.4 Schur product

The Schur Product (denoted by $\odot$) of two $m \times n$ matrices A and B is defined as follows:

$$(A \odot B)_{ij} = A_{ij} B_{ij}.$$

In other words, it is the component-wise product. In many applications we are faced with computing the sequentially semi-separable structure of the Schur product of a sequentially semi-separable matrix and a low-rank matrix. Due to section 10.2 it is enough to look at the Schur product of a sequentially semi-separable matrix with a rank-1 matrix.

We begin with the Schur product of a rank-1 matrix:

$$(A \odot uv^H)_{ij} = u_i A_{ij} v_j.$$

Hence we have that

$$A \odot uv^H = \text{diag}(u) A \text{diag}(v). \tag{10}$$

Now let A be a sequentially semi-separable matrix and let $B = A \odot rs^H$. We would like to compute the sequentially semi-separable representation of B from that of A. This is based on the formulas used to compute the sequentially semi-separable representation in section 6.

First some notation. Let

$$r = \begin{array}{c} m_1 \\ \vdots \\ m_n \end{array} \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix} \quad \text{and} \quad s = \begin{array}{c} m_1 \\ \vdots \\ m_n \end{array} \begin{pmatrix} s_1 \\ \vdots \\ s_n \end{pmatrix}.$$

Then it follows readily from equation (10) that

$$\begin{aligned} U_i(B) &= \operatorname{diag}(r_i)U_i(A), \quad V_i(B) = \operatorname{diag}(s_i)V_i(A), \quad W_i(B) = W_i(A), \quad D_i(B) = D_i(A) \odot (r_i s_i^H), \\ P_i(B) &= \operatorname{diag}(r_i)P_i(A), \quad Q_i(B) = \operatorname{diag}(s_i)Q_i(A), \quad R_i(B) = R_i(A). \end{aligned}$$

10.5 Tensor product

Equally important in applications is the tensor product (denoted by $\otimes$) of a sequentially semi-separable matrix with the identity. The tensor product of an $m_A \times n_A$ matrix A with an $m_B \times n_B$ matrix B is defined as the $m_A m_B \times n_A n_B$ matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n_A}B \\ \vdots & \cdots & \vdots \\ a_{m_A 1}B & \cdots & a_{m_A n_A}B \end{pmatrix}.$$

We observe that this form is preserved under block-partitioning of the first matrix:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \otimes B = \begin{pmatrix} A_{11} \otimes B & A_{12} \otimes B \\ A_{21} \otimes B & A_{22} \otimes B \end{pmatrix}.$$

We also observe that for the tensor product of rank-1 matrices with the identity we have

$$uv^H \otimes I_N = (u \otimes I_N)(v \otimes I_N)^H.$$

Hence it follows that the tensor product of a rank-k matrix with the identity is given by

$$AB^H \otimes I_N = (A \otimes I_N)(B \otimes I_N)^H.$$

We are now interested in computing the sequentially semi-separable structure of $B = A \otimes I_N$ when A is a sequentially semi-separable matrix. To do that, we use the formulas in section 6 together with the formulas just stated about tensor products, to observe that

$$\begin{aligned} U_i(B) &= U_i(A) \otimes I_N, \quad V_i(B) = V_i(A) \otimes I_N, \quad W_i(B) = W_i(A) \otimes I_N, \\ D_i(B) &= D_i(A) \otimes I_N, \quad P_i(B) = P_i(A) \otimes I_N, \quad Q_i(B) = Q_i(A) \otimes I_N, R_i(B) = R_i(A) \otimes I_N. \end{aligned}$$

10.6 Fast Model Reduction

As it became evident in sections 10.2, 10.3, 10.4 and 10.5 sometimes we produce sequentially semi-separable representations that are not as compact as possible (to some specified tolerance τ). Dewilde and van der Veen [14] present a technique to find the optimal reduced order model. Here we present a simple, efficient and numerically stable method to compress a given SSS representation to a pre-specified tolerance τ.

We will present the technique using SVDs. In particular we will call the SVD from which all singular values that are less than τ have been discarded as a τ-accurate SVD. It is a simple matter to replace the SVD in these calculations with rank-revealing QR factorizations, or rank-revealing LU factorizations, or even LU factorization with complete pivoting. It is likely that this will lead to considerable speed-up for a small loss in compression.

Our algorithm for model reduction can be viewed as a natural extension of the algorithm presented in section 6 for constructing SSS representations. More formally, suppose we have a nominal SSS representation for the matrix A. Given a tolerance τ, we would like to compute a new SSS representation for A, accurate to this new tolerance, but more rapidly than the algorithm presented in section 6.

First we observe that it is enough to specify the method for U_i, W_i and V_i, since the same technique can then be applied to the P_i, R_i amd Q_i. We split the method into two stages. In the first stage we convert the representation into **left proper form**. By left proper form we mean that all the column bases C_i of the Hankel-blocks, where

$$\begin{aligned} C_1 &= U_1, \\ C_i &= \begin{pmatrix} C_{i-1}W_i \\ U_i \end{pmatrix}, \end{aligned}$$

should have orthonormal columns. In the second stage we convert the representation into **right proper form**; that is, now all the row bases G_i of the Hankel-blocks, where

$$\begin{aligned} G_n &= V_n, \\ G_i &= \begin{pmatrix} V_i \\ G_{i+1}W_i^H \end{pmatrix}. \end{aligned}$$

will have orthonormal columns. The second stage recursions will essentially be first-stage recursions in the opposite order. Note that $H_i = C_i G_{i+1}^H$.

Note that the method of section 6 already produces SSS representations in left proper form. However it is likely that updating operations will destroy proper form. So we begin by indicating how left proper form can be restored efficiently. For convenience we use hats to denote the representation in left proper form.

Consider the following recursions

$$\begin{aligned} \begin{pmatrix} \tilde{W}_i \\ U_i \end{pmatrix} &\approx \begin{pmatrix} \hat{W}_i \\ \hat{U}_i \end{pmatrix} \Sigma_i F_i^H, \qquad \tau\text{-accurate SVD factorization} \\ \tilde{W}_{i+1} &= \Sigma_i F_i^H W_{i+1}, \\ \hat{V}_{i+1} &= V_{i+1} F_i \Sigma_i^H, \end{aligned}$$

with the understanding that $\tilde{W}_1$ and $\hat{W}_1$ are empty matrices. Then it is easy to check that new column bases

$$\begin{aligned} \hat{C}_1 &= \hat{U}_1 \\ \hat{C}_i &= \begin{pmatrix} \hat{C}_{i-1}\hat{W}_i \\ \hat{U}_i \end{pmatrix}, \end{aligned}$$

have orthonormal columns, and that the hatted sequences form a valid SSS representation for the given matrix. The hatted SSS representation will be accurate to τ provided we assume that the two-norm of the G_i's is bounded by a small multiple of the norm of the original matrix. We will call such SSS representations to be **stable**, and observe that all SSS representations we have produced so far are stable. Also note that the recursions depend only linearly on the matrix size. If by some chance the SSS representation is unstable, then τ must be set to zero in the first stage. It should then be restored to its proper value in the second stage.

In the next stage we take an SSS representation in left proper form and further compress it to the given tolerance τ efficiently by converting it into right proper form. For simplicity we assume that the given

SSS representation is already in left proper form, and denote it using un-hatted quantities. We use hatted quantities to denote terms in the compressed right proper form representation. In the second stage it is sufficient to concentrate on the row bases G_i since $H_i = C_i G_{i+1}^H$, and C_i has orthonormal columns by our assumption of left proper form.

However, this time we must run the recursions backward in time. Here they are

$$\begin{aligned}
\begin{pmatrix} V_i \\ \tilde{W}_i^H \end{pmatrix} &\approx \begin{pmatrix} \hat{V}_i \\ \hat{W}_i^H \end{pmatrix} \Sigma_i F_i^H, && \tau\text{-accurate SVD factorization,} \\
\tilde{W}_{i-1} &= W_{i-1} F_i \Sigma_i^H, \\
\hat{U}_{i-1} &= U_{i-1} F_i \Sigma_i^H,
\end{aligned}$$

with the understanding that $\tilde{W}_n$ and $\hat{W}_n$ are empty matrices. It can be seen that the hatted sequences form a valid SSS representation in right proper form, and that the approximations are τ accurate. We draw attention to the fact that in each stage the W_i matrices are transformed twice; first to $\tilde{W}_i$ and then to $\hat{W}_i$.

11 Superfast solver

We now consider the superfast solution of the linear system of equations $AX = B$, where A and B are given in SSS form, and we desire to find the SSS form of X. One possible approach to the problem is to first compute explicitly the SSS form of U, L and V in the decomposition $A = ULV^H$ and then use the superfast multiplication algorithm. Such an approach has already been described in [14]. Here we show how to extend the implicit decomposition ideas in the same direction.

We will assume that A, X and B are conformally partitioned. It is useful to allow B and X to be non-square matrices. To facilitate this description we will assume that the row and column partitions of the matrix A are indexed from 1 to n. So the SSS representation of A is given by $\{D_i(A)\}_{i=1}^n$, $\{U_i(A)\}_{i=1}^n$, and so on. We will assume that the row partitions of both X and B are also indexed from 1 to n. However, we will assume that the column partitions of X and B are indexed from m to r. Hence the SSS representation for B for example will be given by $\{D_i(B)\}_{i=1}^n$, $\{U_i(B)\}_{i=1}^n$, $\{W_i(B)\}_{i=1}^r$, $\{V_i(B)\}_{i=1}^r$, $\{P_i(B)\}_{i=1}^n$, $\{R_i(B)\}_{i=m}^n$, $\{Q_i(B)\}_{i=m}^n$. Similar considerations hold for X. Throughout this presentation we will pretend that $m \leq 1 \leq n \leq r$. However these assumptions can be removed.

To ease presentation we will assume that the initial equation $AX = B$ is modified to read

$$AX = B - \begin{pmatrix} 0 \\ P_2(A) \\ P_3(A)R_2(A) \\ \vdots \\ P_n(A)R_{n-1}(A)\cdots R_2(A) \end{pmatrix} \begin{pmatrix} Q_m(B)R_{m+1}^H(B)\cdots R_0^H(B)P_1^H(\tau) \\ \vdots \\ Q_0(B)P_1^H(\tau) \\ D_1^H(\tau) \\ V_2(B)U_1^H(\tau) \\ \vdots \\ V_r(B)W_{r-1}^H(B)\cdots W_2^H(B)U_1^H(\tau) \end{pmatrix}^H, \tag{11}$$

where $D_1(\tau) = 0$, $P_1(\tau) = 0$ and $U_1(\tau) = 0$. The reason for this peculiar form will become clear shortly.

The superfast solver is essentially of the same form as the fast solver presented in section 5. The superfast solver can also be presented in a recursive fashion, which we proceed to do.

11.1 Case of $n > 1$ and $k_1(A) < m_1$: Elimination

As before we perform orthogonal eliminations by computing QL and LQ factorizations

$$U_1(A) = q_1 \begin{array}{cc} \begin{pmatrix} 0 \\ \hat{U}_1(A) \end{pmatrix} & \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \end{array},$$

and

$$\left(q_1^H D_1(A)\right) = \begin{array}{l} \\ m_1 - k_1(A) \\ k_1(A) \end{array} \begin{array}{cc} m_1 - k_1(A) & k_1(A) \\ \left(\begin{array}{c} D_{11}(A) \\ D_{21}(A) \end{array}\right. & \left.\begin{array}{c} 0 \\ D_{22}(A) \end{array}\right) \end{array} w_1,$$

where q_1 and w_1 are unitary matrices. We now need to apply q_1^H to the first m_1 rows of equation (11). As before nothing needs to be done for A, X and the second term involving τ on the right-hand side. Applying q_1^H to the first m_1 rows of B we modify $P_1(B)$, $D_1(B)$ and $U_1(B)$ as follows:

$$\begin{aligned}
q_1^H P_1(B) &= \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{P}_1(B) \\ \hat{P}_1(B) \end{pmatrix}, \\
q_1^H D_1(B) &= \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{D}_1(B) \\ \tilde{D}_2(B) \end{pmatrix}, \\
q_1^H U_1(B) &= \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{U}_1(B) \\ \hat{U}_1(B) \end{pmatrix}.
\end{aligned}$$

Next we need to apply w_1 to the first m_1 rows of X. Of course this is a purely formal process since the first m_1 rows of X are unknown at this stage. Here are the quantities that are modified:

$$\begin{aligned}
w_1 P_1(X) &= \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{P}_1(X) \\ \hat{P}_1(X) \end{pmatrix}, \\
w_1 D_1(X) &= \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{D}_1(X) \\ \hat{D}_1(X) \end{pmatrix}, \\
w_1 U_1(X) &= \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{U}_1(X) \\ \hat{U}_1(X) \end{pmatrix}.
\end{aligned}$$

We also need to apply w_1^H to the first m_1 columns of the coefficient matrix. Since it has already been applied to $D_1(A)$, we only need to compute

$$w_1 Q_1(A) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{Q}_1(A) \\ \hat{Q}_1(A) \end{pmatrix}.$$

We now observe that the first $m_1 - k_1(A)$ equations of the transformed system are of the form

$$D_{11}(A)\begin{pmatrix} Q_m^H(X)R_{m+1}^H(X)\cdots R_0^H(X)\tilde{P}_1^H(X) \\ \vdots \\ Q_0^H(X)\tilde{P}_1^H(X) \\ \tilde{D}_1^H(X) \\ V_2(X)\tilde{U}_1^H(X) \\ \vdots \\ V_r(X)W_{r-1}^H(X)\cdots W_2^H(X)\tilde{U}_1^H(X) \end{pmatrix}^H = \begin{pmatrix} Q_m^H(B)R_{m+1}^H(B)\cdots R_0^H(B)\tilde{P}_1^H(B) \\ \vdots \\ Q_0^H(B)\tilde{P}_1^H(B) \\ \tilde{D}_1^H(B) \\ V_2(B)\tilde{U}_1^H(B) \\ \vdots \\ V_r(B)W_{r-1}^H(B)\cdots W_2^H(B)\tilde{U}_1^H(B) \end{pmatrix}^H.$$

Solving this we obtain

$$\begin{aligned} \tilde{D}_1(X) &= D_{11}^{-1}(A)\tilde{D}_1(B), \\ \tilde{P}_1(X) &= (\, D_{11}^{-1}(A)\tilde{P}_1(B) \quad 0\,), \\ Q_i(X) &= \begin{pmatrix} Q_i(B) & \vec{Q}_i \end{pmatrix}, \\ R_i(X) &= \begin{pmatrix} R_i(B) & 0 \\ \vec{R}_{i;21} & \vec{R}_{i;22} \end{pmatrix}, \\ \tilde{U}_1(X) &= (\, D_{11}^{-1}(A)\tilde{U}_1(B) \quad 0\,), \\ V_i(X) &= \begin{pmatrix} V_i(B) & \vec{V}_i \end{pmatrix}, \\ W_i(X) &= \begin{pmatrix} W_i(B) & 0 \\ \vec{W}_{i;21} & \vec{W}_{i;22} \end{pmatrix}. \end{aligned}$$

We now need to subtract from the right-hand side the first $m_1 - k_1(A)$ columns of the coefficient matrix multiplied by the first $m_1 - k_1(A)$ rows of the transformed unknowns. We first subtract $D_{21}(A)$ times the first $m_1 - k_1(A)$ rows of the unknown from the corresponding rows of B. We observe that it leads to the following changes:

$$\begin{aligned} \hat{D}_1(B) &= \tilde{D}_2(B) - D_{21}(A)D_{11}^{-1}(A)\tilde{D}_1(B) = \tilde{D}_2(B) - D_{21}(A)\tilde{D}_1(X), \\ \hat{U}_1(B) &= \tilde{U}_2(B) - D_{21}(A)D_{11}^{-1}(A)\tilde{U}_1(B), \\ \hat{P}_1(B) &= \tilde{P}_2(B) - D_{21}(A)D_{11}^{-1}(A)\tilde{P}_1(B). \end{aligned}$$

To subtract the remaining rows of the first $m_1 - k_1(A)$ columns of the coefficient matrix from the right-hand side, we observe that it can be merged with the τ terms as follows:

$$\begin{aligned} \hat{D}_1(\tau) &= D_1(\tau) + \tilde{Q}_1^H(A)D_{11}^{-1}(A)\tilde{D}_1(B), \\ \hat{P}_1(\tau) &= P_1(\tau) + \tilde{Q}_1^H(A)D_{11}^{-1}(A)\tilde{P}_1(B), \\ \hat{U}_1(\tau) &= U_1(\tau) + \tilde{Q}_1^H(A)D_{11}^{-1}(A)\tilde{U}_1(B). \end{aligned}$$

If we now discard the first $m_1 - k_1(A)$ rows and columns we are left with a set of equations that are identical in structure to the system (11) that we started with. In particular we replace in the SSS form for A, X and B, all terms with subscript 1 by their corresponding hatted forms. The resulting hatted system $\hat{A}\hat{X} = \hat{B} - (\tau \text{ terms})$, looks exactly like the equation (11) we started with and can be solved recursively for $\hat{X}$. Once we have $\hat{X}$ we need to recover X. The right formulas are:

$$\begin{aligned} D_1(X) &= w_1^H \begin{pmatrix} \tilde{D}_1(X) \\ \hat{D}_1(X) \end{pmatrix}, \\ U_1(X) &= w_1^H \begin{pmatrix} (\, D_{11}^{-1}(A)\tilde{U}_1(B) \quad 0\,) \\ (\, \hat{U}_1(X) \quad 0\,) \end{pmatrix}, \\ W_2(X) &= \begin{pmatrix} W_2(B) & 0 \\ \hat{W}_{2;1}(X) & \hat{W}_{2;2}(X) \end{pmatrix}, \end{aligned}$$

$$\begin{aligned}
V_2(X) &= (\, V_2(B) \quad \hat{V}_2(X) \,), \\
P_1(X) &= w_1^H \begin{pmatrix} (\, D_{11}^{-1}(A)\tilde{P}_1(B) \quad 0 \,) \\ (\, \hat{P}_1(X) \quad 0 \,) \end{pmatrix}, \\
R_2(X) &= \begin{pmatrix} R_2(B) & 0 \\ \hat{R}_{2;1}(X) & \hat{R}_{2;2}(X) \end{pmatrix}, \\
Q_i(X) &= \hat{Q}_i(X),
\end{aligned}$$

where

$$\begin{aligned}
\begin{pmatrix} \hat{W}_2(X) \\ 0 \end{pmatrix} &= (\, \hat{W}_{2;1}(X) \quad \hat{W}_{2;2}(X) \,), \\
\begin{pmatrix} \hat{R}_2(X) \\ 0 \end{pmatrix} &= (\, \hat{R}_{2;1}(X) \quad \hat{R}_{2;2}(X) \,).
\end{aligned}$$

These formulas can be verified only by looking at the next two cases.

11.2 Case for $n > 1$ and $k_1(A) \geq m_1$: Merge

The second possibility is that the $k_1(A)$ is not large enough to permit efficient elimination. In this case we merge the the first two block rows of the equation. We also merge the first two block columns of A. However for X and B we do not merge any block columns. Rather we move the diagonal block over by one position to the right. For convenience we specify the quantities that change for each of A, X and B. Beginning with A we have:

$$\begin{aligned}
\hat{D}_1(A) &= \begin{pmatrix} D_1(A) & U_1(A)V_2^H(A) \\ P_2(A)Q_1^H(A) & D_2(A) \end{pmatrix}, \\
\hat{U}_1(A) &= \begin{pmatrix} U_1(A)W_2(A) \\ U_2(A) \end{pmatrix}, \\
\hat{Q}_1(A) &= \begin{pmatrix} Q_1(A)R_2^H(A) \\ Q_2(A) \end{pmatrix}.
\end{aligned}$$

Next we need to merge the first two block rows of B and move the diagonal block one over to the right:

$$\begin{aligned}
\tilde{D}_1(B) &= \begin{pmatrix} U_1(B)V_2^H(B) \\ D_2(B) \end{pmatrix}, \\
\hat{D}_i(B) &= D_{i+1}(B), \quad i = 2, \ldots, n-1, \\
\tilde{P}_1(B)\hat{Q}_0(B) &= \begin{pmatrix} P_1(B) & D_1(B) \\ P_2(B)R_1(B) & P_2(B)Q_1^H(B) \end{pmatrix}, \qquad \tau\text{-accurate low-rank factorization,} \\
\tilde{P}_1(B) &= \begin{pmatrix} P_1(B) & \vec{P}_1(B) \\ P_2(B)R_1(B) & \vec{P}_2(B) \end{pmatrix}, \\
\hat{P}_i(B) &= P_{i+1}(B), \quad n = 2, \ldots, n-1, \\
\hat{R}_0(B) &= \begin{pmatrix} I \\ 0 \end{pmatrix}, \\
\hat{R}_i(B) &= R_{i+1}(B), \quad i = -1, \ldots, m-1, \\
\hat{R}_1(B) &= (\, R_1(B) \quad 0 \,), \\
\hat{R}_i(B) &= R_i(B), \quad i = 2, \ldots, r \\
\hat{Q}_i(B) &= Q_{i+1}(B), \quad n = -1, \ldots, m-1, \\
\hat{Q}_0(B) &= \big(\, Q_1(B) \quad \vec{Q}_1(B) \,\big), \\
\hat{Q}_i(B) &= Q_{i+1}, \quad i = 1, \ldots, r-1,
\end{aligned}$$

$$
\begin{aligned}
\tilde{U}_1(B) &= \begin{pmatrix} U_1(B)W_2(B) \\ U_2(B) \end{pmatrix}, \\
\hat{U}_i(B) &= U_{i+1}(B), \quad i = 2, \ldots, n-1,
\end{aligned}
$$

where $\vec{P}_1(B)$, $\vec{P}_2(B)$ and $\vec{Q}_1(B)$ are chosen further down. We just note as a warning that $\tilde{P}_1(B)\hat{Q}_0^H(B)$ is *not* conformally partitioned in the above formulas.

Some quantities in the SSS representation for B still do not wear hats since we still need to subtract the second block row of the τ terms from B. To do that we must first move the diagonal τ block one position over to the right. Doing that we obtain

$$
\begin{aligned}
\tilde{D}_1(\tau) &= U_1(\tau)V_2^H(B), \\
\tilde{P}_1(\tau) &= \begin{pmatrix} P_1(\tau) & \vec{P}_1(\tau) \end{pmatrix}, \\
\tilde{U}_1(\tau) &= U_1(\tau)W_2(B),
\end{aligned}
$$

where we pick $\vec{P}_1(\tau)$, $\vec{P}_1(B)$, $\vec{P}_2(B)$ and $\vec{Q}_1(B)$ such that

$$
\begin{pmatrix} \begin{pmatrix} D_1(B) \\ P_2(B)Q_1^H(B) \end{pmatrix} \\ D_1(\tau) \end{pmatrix} \approx \begin{pmatrix} \tilde{P}_1(B) \\ \tilde{P}_1(\tau) \end{pmatrix} \hat{Q}_0^H(B).
$$

To do this efficiently the method of section **??** must be suitably modified.

Now we can subtract the second block row of the τ terms from the new first block of B. The corresponding changes to B are:

$$
\begin{aligned}
\hat{P}_1(B) &= \tilde{P}_1(B) - \begin{pmatrix} 0 \\ P_2(A) \end{pmatrix} \tilde{P}_1(\tau), \\
\hat{U}_1(B) &= \tilde{U}_1(B) - \begin{pmatrix} 0 \\ P_2(A) \end{pmatrix} \tilde{U}_1(\tau), \\
\hat{D}_1(B) &= \tilde{D}_1(B) - \begin{pmatrix} 0 \\ P_2(A) \end{pmatrix} \tilde{D}_1(\tau).
\end{aligned}
$$

We must now return the τ terms to canonical form with the first two block rows being merged to zero. The changes are:

$$
\begin{aligned}
\hat{P}_1(\tau) &= R_2(A)\tilde{P}_1(\tau), \\
\hat{D}_1(\tau) &= R_2(A)\tilde{D}_1(\tau), \\
\hat{U}_1(\tau) &= R_2(A)\tilde{U}_1(\tau).
\end{aligned}
$$

Now the hatted sequences represent an SSS system with $n-1$ block rows. This can be solved recursively for the hatted SSS representation for X. From that we must recover the original SSS representation for X involving n row partitions. To do that we observe that we need to split the first block row of the hatted representation and shift the first diagonal block one position to the left. We begin by first splitting the first block row:

$$
\begin{aligned}
\hat{D}_1(X) &= \begin{pmatrix} \tilde{D}_1(X) \\ D_2(X) \end{pmatrix}, \\
\hat{U}_1(X) &= \begin{pmatrix} \tilde{U}_1(X) \\ U_2(X) \end{pmatrix}, \\
\hat{P}_1(X) &= \begin{pmatrix} \tilde{P}_1(X) \\ P_2(X) \end{pmatrix}.
\end{aligned}
$$

The rest of the changes to X are

$$\begin{aligned}
D_1(X) &= \tilde{P}_1(X)\hat{Q}_0^H(X), \\
P_1(X) &= \tilde{P}_1(X)\hat{R}_0(X), \\
P_i(X) &= \hat{P}_{i-1}(X), \quad i = 3, \ldots, n, \\
R_i(X) &= \hat{R}_{i-1}(X), \quad i = m, \ldots, n, \\
Q_i(X) &= \hat{Q}_{i-1}(X), \quad i = m, \ldots, n, \\
U_1(X) &= \begin{pmatrix} \tilde{U}_1(X) & \vec{U}_1(X) \end{pmatrix}, \\
W_2(X) &= \begin{pmatrix} I \\ 0 \end{pmatrix}, \\
W_i(X) &= \hat{W}_{i-1}(X), \quad i = 3, \ldots, n, \\
V_i(X) &= \hat{V}_{i-1}(X), \quad i = 3, \ldots, n,
\end{aligned}$$

where $V_2(X)$ and $\vec{U}_1(X)$ are chosen such that

$$\tilde{D}_1(X) \approx U_1(X)V_2^H(X).$$

11.3 Case for $n = 1$: Base

In this case the equations can be solved directly to determine the SSS form for X as follows:

$$\begin{aligned}
D_1(X) &= D_1^{-1}(A)D_1(B), \\
P_1(X) &= D_1^{-1}(A)P_1(B), \\
R_i(X) &= R_i(B), \\
Q_i(X) &= Q_i(B), \\
U_1(X) &= D_1^{-1}(A)U_1(B), \\
W_i(X) &= W_i(B), \\
V_i(X) &= V_i(B).
\end{aligned}$$

Note that the τ terms are just zeros.

12 Applications

We now apply the techniques of this paper to construct fast direct solvers for the first time for two applications that are essentially global spectral methods for differential equations. The first is by Greengard and Rokhlin [23, 31, 37, 38] for boundary-value problems for ordinary differential equations, and the second is by Kress [12, 30, 32] for 2D exterior scattering problems.

12.1 Two-point BVP

Consider the system of differential equations

$$\begin{aligned}
a(t)\frac{dx(t)}{dt} + b(t)x(t) &= f(t), \qquad 0 \le t \le 1, \\
c_0 x(0) + c_1 x(1) &= z,
\end{aligned}$$

	size					
tolerance	256	512	1024	2048	4096	8192
1E-8	9	9	10	10	10	10
1E-12	13	15	16	17	18	18

Table 3: Peak Hankel block ranks for the spectral method of Greengard and Rokhlin for ODEs.

where $x(t)$ and $f(t)$ are N-dimensional vector-valued functions of t; $a(t)$ and $b(t)$ are $N \times N$-matrix valued functions of t; c_0 and c_1 and $N \times N$ matrices; and z is an N-dimensional vector. Then the p-th order Greengard-Rokhlin [23] method produces matrices of the form

$$G = \mathrm{diag}(A_i) + \mathrm{diag}(P_i)(S_f \otimes I_N)\mathrm{diag}(Q_i) + \mathrm{diag}(U_i)(S_b \otimes I_N)\mathrm{diag}(V_i),$$

where S_f and S_b are the p-th order forward and backward Chebyshev spectral integration operators, and A_i, P_i, Q_i, U_i and V_i are $N \times N$ matrices that depend on the coefficients of the ordinary differential equation. The spectral integration operators are given more specifically by the following formulas. Let

$$\begin{aligned} V_{ij} &= T_{i-1}(z_{j-1}^{(p)}), \qquad 1 \le i, j \le p, \\ W_{ij} &= T_{i-1}(z_{j-1}^{(p+1)}), \qquad 1 \le i \le p, \quad 1 \le j \le p+1. \end{aligned}$$

where $z_j^{(p)}$ is the j-th zero of the p-th Chebyshev polynomial T_p. Let S denote the $(p+1) \times p$ indefinite integration operator

$$S_{ij} = \begin{cases} 0, & \text{if } i = 1 \text{ or } i = j \\ 1, & \text{if } i = 2 \text{ and } j = 1 \\ \dfrac{1}{2(i-1)}, & \text{if } i = j+1 \\ \dfrac{-1}{2(i-1)}, & \text{if } i = j-1 \\ 0, & \text{otherwise} \end{cases}$$

Then we have

$$S_f = W(I - e_1 \begin{pmatrix} 0 & 1 & -1 & 1 & -1 & 1 & \cdots \end{pmatrix})SV^{-1}, \;\; S_b = W(e_1 \begin{pmatrix} 0 & 1 & 1 & 1 & \cdots \end{pmatrix} - I)SV^{-1}.$$

In table 3 we report the maximum ranks of the Hankel blocks of S_f and S_b. The rows are indexed by the (absolute) tolerance we used in computing the ranks of the Hankel-blocks. We used two tolerances: 10^{-8} and 10^{-12}. These are the most representative for use in practice. The columns are indexed by the size of the matrices. We report on sizes that range from 256 to 8192. It is clear that for the reported tolerances (10^{-8} and 10^{-12}) the Hankel-block ranks are reasonably small. In fact they seem to behave like $O(\log N)$, where N is the size of the matrix. This implies that this class of matrices can be inverted stably in $O(N \log^2 N)$ flops.

In this example we observe that the sequentially semi-separable representations for S_f and S_b need to be calculated once and stored for each size. From that the sequentially semi-separable representation for G itself can be computed rapidly on the fly.

12.2 Two-dimensional Scattering

For two-dimensional exterior scattering problems on analytic curves for acoustic and electro-magnetic waves, Kress' method of discretization of order $2n$ will lead to a $2n \times 2n$ matrix of the form

$$A = I + R \odot K_1 + K_2,$$

	size					
tolerance	256	512	1024	2048	4096	8192
1E-8	28	32	34	37	38	40
1E-12	40	46	52	58	62	66

Table 4: Peak Hankel block ranks for the spectral method of Kress, Martensen and Kussmaul for the exterior Helmholtz problem.

where K_1 and K_2 are low-rank matrices and

$$R_{ij} = -\frac{2\pi}{n} \sum_{m=1}^{n-1} \frac{1}{m} \cos \frac{m|i-j|\pi}{n} - \frac{(-1)^{|i-j|}\pi}{n^2}.$$

From the results in section 10 we see that it is sufficient to verify that R is a sequentially semi-separable matrix of low Hankel-block ranks. It would then follow that A is a sequentially semi-separable matrix of low Hankel-block ranks. In Table 4 we exhibit the peak Hankel block ranks of R. The rows are indexed by the (absolute) tolerance we used to determine the numerical ranks of the Hankel blocks. In particular we used tolerances of 10^{-8} and 10^{-12} that are useful in practice. The columns are indexed by the size of R.

As can be seen the ranks seem to depend logarithmically on the size N, of R. This implies that the fast algorithm will take $O(N \log^2 N)$ flops to solve linear systems involving A. We observe that the sequentially semi-separable representations of R for different sizes and tolerances need to be computed once and stored off-line. Then using the results in section 10 we can compute the sequentially semi-separable representation of A rapidly on the fly.

13 Related and Future Work

It is a rather difficult job to attribute due credit to all the researchers whose work is related to this paper. Hence we will just mention as much of the related recent literature that we are aware off. The algebraic formalism of the structure that is used in this paper appears in work of Dewilde and van der Veen [14] and Eidelman and Gohberg [17, 18, 19]. Other structures that would work just as well appear in the works of Rokhlin [37, 38] and Hackbusch [26, 27, 28] and Chandrasekaran and Gu [6]. Fast, direct, but not necessarily stable algorithms for such matrices were first presented in Starr's thesis [37, 38]. They also appear in the works of Eidelman and Gohberg [17, 18] and Hackbusch [26, 27, 28]. Fast and stable direct solvers appeared first in the works of Dewilde and van der Veen [14] and independently later in the work of Chandrasekaran and Gu [5]. Various generalizations have been carried out by various authors, including us. Here is a brief list of such work: [7, 8, 9, 27, 28, 33, 34].

The computational electromagnetics literature has also looked at this problem independently. Relevant work includes that of Chew [25], Canning [3], and Jandhyala [21]. It is our hope that this paper would provide a unifying point of view to the many disparate attempts available in the computational electromagnetics literature.

In this paper we presented a fast and stable direct method for solving a linear system of equations whose coefficient matrix satisfies the sequentially semi-separable matrix structure as defined in equation (1), and we demonstrated that many dense matrices coming from various physical problems can be well-approximated by matrices with such structures, thereby significantly reducing the solution time for related linear systems of equations.

The sequentially semi-separable representation used in this paper is not optimal. For example none of the matrices U_i, W_i, V_i, P_i, Q_i, or R_i need be full-rank matrices. In such a case it is better to allow these matrices to be represented in compressed form. This will cause some complications in the algorithms, but nothing major.

It is easy to replace the QL and LQ factorizations with LU factorizations with partial pivoting instead. Such an approach would still be numercially stable as long as the integers m_i's are not very large.

The fast multipole method (FMM) of Greengard and Rokhlin [4, 22, 36] has evolved into a major workhorse for many computationally intensive problems in a wide area of applications. It turns out that the matrix structure exploited by the FMM to speed up the computation is closely related to the sequentially semi-separable structure exploited in this paper. Future work includes developing fast direct solvers for linear systems of equations where the coefficient matrix has the FMM matrix structure.

We will also explore the applications of the fast sequentially semi-separable algorithms to speed up Kress' global discretization technique for solving the integral equations of scattering theory [11].

Acknowledgements. The authors are grateful to Robert Sharpe and Daniel White of Lawrence Livermore National Laboratory for their suggestions and interests in this research.

References

[1] D. Alpay and P. Dewilde, Time-varying signal approximation and estimation, *vol. III of Proc. Int. Symposium MTNS-89*, pp. 1-22, Birkhäuser Verlag, 1990.

[2] D. Alpay, P. Dewilde and H. Dym, Lossless inverse scattering and reproducing kernels for upper triangular matrices, *Operator Theory Advances and Applications,* pp. 61-135, Birkhäuser Verlag 1990.

[3] F. X. Canning and K. Rogovin, *Fast direct solution of moment-method matrices*, IEEE Antennas and Propagation Magazine, 40 (1998).

[4] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 669–686.

[5] S. Chandrasekaran and M. Gu, *Fast and stable algorithms for banded plus semi-separable matrices*, submitted to SIAM J. Matrix Anal. Appl., 2000.

[6] S. Chandrasekaran and M. Gu, *A fast and stable solver for recursively semi-separable systems of equations*, in *Structured matrices in mathematics, computer science and engineering, II*, edited by Vadim Olshevsky, in the Contemporary Mathematics series, AMS publications, 2001.

[7] S. Chandrasekaran and M. Gu, *Fast and Stable Eigendecomposition of Symmetric Banded plus Semi-separable Matrices*, 1999, *Linear Algebra and its Applications*, Volume 313, Issues 1-3, 1 July 2000, pages 107-114.

[8] S. Chandrasekaran and M. Gu, *A Divide-and-Conquer Algorithm for the Eigendecomposition of Symmetric Block-Diagonal Plus Semiseparable Matrices*, 1999, accepted for publication in *Numerische Mathematik.*

[9] S. Chandrasekaran, M. Gu, and T. Pals, *A fast and stable solver for smooth recursively semi-separable systems.* Paper presented at the SIAM Annual Conference, San Diego, CA, 2001, and SIAM Conference of Linear Algebra in Controls, Signals and Systems, Boston, MA, 2001.

[10] S. Chandrasekaran and I. Ipsen, *On Rank-revealing QR Factorizations*, SIAM J. Sci. Comput., 1994, vol. 15, pages 592-622.

[11] D. Colton and R. Kress, *Integral Equation Methods in Scattering Theory*, Wiley, 1983.

[12] D. Colton and R. Kress, *Inverse acoustic and electromagnetic scattering theory,* Applied Mathematical Sciences, vol. 93, Springer-Verlag, 1992.

[13] P. Dewilde and A.-J. van der Veen. *Time-varying Systems and Computations.* Kluwer, 1998.

[14] P. Dewilde and A. van der Veen, *Time-varying systems and computations.* Kluwer Academic Publishers, 1998.

[15] P. Dewilde and A.J. van der Veen. Inner-outer factorization and the inversion of locally finite systems of equations. *Linear Algebra and its Applications*, 313:53–100, 2000.

[16] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, *Algorithm 679: A Set of Level 3 Basic Linear Algebra Subprograms: Model Implementation and Test Programs*, ACM Transactions on Mathematical Software, 16 (1990), pp. 18-28.

[17] Y. Eidelman and I. Gohberg, *Inversion formulas and linear complexity algorithm for diagonal plus semiseparable matrices*, Computers and Mathematics with Applications, 33 (1997), Elsevier, pp. 69–79.

[18] Y. Eidelman and I. Gohberg, *A look-ahead block Schur algorithm for diagonal plus semiseparable matrices*, Computers and Mathematics with Applications, 35 1998), pp. 25–34.

[19] Y. Eidelman and I. Gohberg, *A modification of the Dewilde van der Veen method for inversion of finite structured matrices,* Linear Algebra and its Applications, volumes 343–344, 1 march 2001, pages 419–450.

[20] I. Gohberg, T. Kailath and I. Koltracht, Linear complexity algorithms for semiseparable matrices, *Integral Equations and Operator Theory*, vol. 8, pp. 780-804, Birkhäuser Verlag, 1985.

[21] D. Gope and V. Jandhyala, *An iteration-free fast multilevel solver for dense method of moment systems,* Electrical Performance of Electronic Packaging, 2001, pp. 177–180.

[22] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), pp. 325–348.

[23] L. Greengard and V.Rokhlin, *On the numerical solution of 2-point boundary value problems*, Communications on Pure and Applied Mathematics, June 1991, vol. 44, no. 4, pages 419–452.

[24] M. Gu and S. C. Eisenstat, *Efficient Algorithms for Computing a Strong Rank-revealing QR Factorization*, SIAM J. Sci. Comput., 1996, vol. 17, pages 848-869.

[25] L. Gurel and W. C. Chew, *Fast direct (non-iterative) solvers for integral-equation formulations of scattering problems,* IEEE Antennas and Propagation Society International Symposium, 1998 Digest, Antennas: Gateways to the Global Network, pp. 298–301, vol. 1.

[26] W. Hackbusch, *A sparse arithmetic based on $\mathcal{H}$-matrices. Part-I: Introduction to $\mathcal{H}$-matrices*, Computing 62, pp. 89–108, 1999.

[27] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part-II: application to multi-dimensional problems*, Computing 64, pp. 21–47, 2000.

[28] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic: general complexity estimates,* Journal of Computational and Applied Mathematics, volume 125, pp. 79–501, 2000.

[29] T. Kailath, Fredholm Resolvents, Wiener-Hopf Equations, and Riccati Differential Equations, *IEEE Trans. on Information Theory,* vol. IT-15, No. 6, Nov, 1969.

[30] R. Kussmaul, *Ein numerisches Verfahren zur Lösung des Neumannschen Aussenraumproblems für die Helmholtzsche Schwingungsgleichung,* Computing 4, 246–273, 1969.

[31] June-Yub Lee and L. Greengard, *A fast adaptive numerical method for stiff two-point boundary value problems,*SIAM Journal on Scientific Computing, vol.18, (no.2), SIAM, March 1997, pp. 403–29.

[32] E. Martensen, *Über eine Methode zum räumlichen Neumannschen Problem mit einer Anwendung für torusartige Berandungen,* Acta Math. 109, 75–135, 1963.

[33] N. Mastronardi, S. Chandrasekaran and S. van Huffel, *Fast and stable two–way chasing algorithm for diagonal plus semi-separable systems of linear equations*, *Numerical Linear Algebra with Applications*, Volume 38, Issue 1, January 2000, pages 7–12.

[34] N. Mastronardi, S. Chandrasekaran and S. van Huffel, *Fast and stable algorithms for reducing diagonal plus semi-separable matrices to tridiagonal and bidiagonal form*, *BIT*, volume 41, number 1, March 2001, pages 149–157.

[35] T. Pals. Ph. D. Thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, 2002.

[36] V. Rokhlin, *Applications of volume integrals to the solution of PDEs*, J. Comp. Phys., 86 (1990), pp. 414–439.

[37] P. Starr, *On the numerical solution of one-dimensional integral and differential equations*, Thesis advisor: V. Rokhlin, Research Report YALEU/DCS/RR-888, Department of Computer Science, Yale University, New Haven, CT, December 1991.

[38] P. Starr and V. Rokhlin, *On the numerical solution of 2-point boundary value problem.2* Communications on Pure and Applied Mathematics, Aug. 1994, vol. 47, no. 8, pages 1117–1159.

[39] A.-J. van der Veen, Time-varying lossless systems and the inversion of large structured matrices, *Archiv f. Elektkronik u. Übertragungstechnik*, 49 (5/6) (1995) pp. 372-382.