SIAM J. MATRIX ANAL. APPL.
Vol. 27, No. 2, pp. 341–364

SOME FAST ALGORITHMS FOR SEQUENTIALLY SEMISEPARABLE REPRESENTATIONS[*]

S. CHANDRASEKARAN[†], P. DEWILDE[‡], M. GU[§], T. PALS[¶], X. SUN[‖], A.-J. VAN DER VEEN[‡], AND D. WHITE[**]

Abstract. An extended sequentially semiseparable (SSS) representation derived from time-varying system theory is used to capture, on the one hand, the low-rank of the off-diagonal blocks of a matrix for the purposes of efficient computations and, on the other, to provide for sufficient descriptive richness to allow for backward stability in the computations. We present (i) a fast algorithm (linear in the number of equations) to solve least squares problems in which the coefficient matrix is in SSS form, (ii) a fast algorithm to find the SSS form of X such that $AX = B$, where A and B are in SSS form, and (iii) a fast model reduction technique to improve the SSS form.

Key words. backward stability, fast direct solver, fast multipole method, integral equations, method of moments, scattering theory, semiseparable matrices, spectral methods, Moore–Penrose inverses, time-varying systems

AMS subject classifications. 65F05, 65F20, 65F30

DOI. 10.1137/S0895479802405884

1. Introduction.

In this paper we will present fast backward stable algorithms for the solution of a large class of least squares problems with structured matrices which can be either sparse or dense. The structure concerned is called "semiseparable" and is a matrix analogue of semiseparable integral kernels as described by Kailath in [33]. This matrix analogue was most likely first described by Gohberg, Kailath, and Koltracht in [22]. In that paper it is shown that, under further technical restrictions, an LDU factorization is possible with a complexity p^2N, where p is the complexity of the semiseparable description and N the dimension of the matrix, in effect an algorithm linear in the size of the matrix, when p 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 analogous 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 [22] and allows for more general types of efficient operations. (By

[*]Received by the editors April 17, 2002; accepted for publication (in revised form) by S. Van Huffel October 25, 2004; published electronically September 15, 2005.
http://www.siam.org/journals/simax/27-2/40588.html

[†]Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106-9560 (shiv@ece.ucsb.edu). This author's research was supported in part by grant B521537 from Lawrence Livermore National Laboratory, NSF Career Award CCR-9734290, and NSF grant CCR-0204388.

[‡]Faculty of Electrical Engineering, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands (p.dewilde@its.tudelft.nl, A.J.vanderVeen@its.tudelft.nl).

[§]Department of Mathematics, University of California at Berkeley, Berkeley, CA 94720-3840 (mgu@math.berkeley.edu). This author's research was supported in part by NSF grant CCR-0204388 and Lawrence Livermore National Laboratory grant ISCR-03-24.

[¶]Department of Electrical and Computer Engineering, University of California at Santa Barbara (tim@kipling.ece.ucsb.edu). This author's research was supported in part by Lawrence Livermore National Laboratory grant ISCR-03-24.

[‖]Department of Mechanical Engineering, University of California at Berkeley (sunx@me.berkeley.edu).

[**]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551 (dwhite@llnl.gov).

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

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

"efficient" we mean operations that are linear in the size of the matrix.) In the book *Time-Varying Systems and Computations* [16], Dewilde and van der 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) semiseparable system can be done with an efficient recursive procedure. This procedure is based on the ideas presented in [40] and then further elaborated by Dewilde and van der Veen in [17] and by Eidelman and Gohberg in [21]. In the former paper the connection with Kalman filtering as a special case of the procedures is also discussed. Other related works include [35, 41].

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

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

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. 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 matrices D_i, U_i, W_i, V_i, P_i, Q_i, and R_i provide a sequentially semiseparable (SSS) representation of the matrix A if they satisfy (1).

In the case where all W_i and R_i are identities, A reduces to a block-diagonal plus semiseparable matrix, which can be handled directly using techniques in Chandrasekaran and Gu [8]. It is shown in [16] that this class of matrices is closed under inversion and includes banded matrices and semiseparable matrices, as well as their inverses as special cases.

It should be noted that the SSS structure of a given matrix A depends on the sequence m_i. Different sequences will lead to different representations.

In this paper we present a fast backward stable algorithm for solving a linear least squares problem where the coefficient matrix is a rectangular matrix with a structure similar to (1). As a consequence, this work effectively shows that the Moore–Penrose inverses of such matrices have similar structures and can be computed rapidly as well. We also present a fast algorithm for computing the SSS representation of X, such that $AX = B$, when the SSS representation of both A and B are known. For both these algorithms we present numerical experiments on timing and accuracy. Finally, we present a fast model reduction algorithm for improving the efficiency of the SSS representation when necessary. This latter part of the paper is, in effect, a generalization of the method presented in [8], which is not repeated here.

A companion technical report [7] is also available and is more comprehensive in the topics it covers.

2. One-pass solution for the Moore–Penrose inverse of a general system.

Given a general matrix A in SSS form and a vector b with the same number of rows as 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 that is one-pass and top-down. As before, we assume that we have a realization consisting of block matrices $\{U_i\}$, $\{V_i\}$, etc., with the only restriction being that these realization matrices must be appropriately sized to fit the products in the representation. In particular, the diagonal entries $\{D_i\}$ do not have to be square or nonsingular. In [16] a general algorithm was given to determine the Moore–Penrose inverse of a general time-varying operator. In this paper, we adapt this algorithm to the matrix case using the ideas of [6]. This results in a fast one-pass and top-down algorithm for the structured least squares problem.

The one-pass character is obtained by making judicious use of left and right multiplications by elementary orthogonal matrices. These matrices then combine to form semiseparable operators of the same complexity as the original semiseparable matrix. What makes the Moore–Penrose case more complex than the square invertible case 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 $x = (b_1 + b_2)/2$; that is, the first equation cannot 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 left null-space has to be implicitly determined (see what follows).

The strategy we follow to obtain the reduction to a row-independent form works in two steps and produces a system of independent rows in block upper triangular form. To find the Moore–Penrose solution a further step then consists in solving the resulting row-independent system using the method described in [6] (this time actually on a simpler system) or just by constructing the inverse directly as is done in [16]. The resulting complexity will roughly be twice that of the direct inversion algorithm, in which invertibility of the matrix is assumed. 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 of complex entries unitary) transforms on the columns (on the right of the matrix), we gradually transform the double-sided representation into a block upper triangular one, that is, one in which the P-, Q-, and R-type matrices are all zero. The block upper triangular representation will have the same complexity as the original. The result is the representation of a block upper triangular matrix A_1, related to

A through

$$A_1 = A\hat{Q},$$

where $\hat{Q}$ assembles the column transformations. (We keep using the symbol "Q" with a qualification such as a hat or a tilde to indicate unitary transformation matrices. These Q's should not be confused with the Q_i in the original representation. A similar remark applies to the use of R's.)

2. Using elementary orthogonal transformations on the rows (i.e., on the left) we gradually (lazily) determine the left null-space and range of A_1:

$$A_1 = [\,\tilde{Q}^{(1)} \quad \tilde{Q}^{(2)}\,] \begin{bmatrix} 0 \\ A_o \end{bmatrix},$$

in which A_o has linearly independent rows. The columns of $\tilde{Q}^{(2)}$ then form an orthonormal basis for the range of A_1 and the columns of $\tilde{Q}^{(1)}$ for its left null-space (i.e., the kernel of A_1^H).

3. The combination of the two previous steps has produced

$$A = [\,\tilde{Q}^{(1)} \quad \tilde{Q}^{(2)}\,] \begin{bmatrix} 0 \\ A_o \end{bmatrix} \hat{Q}.$$

With $y = \hat{Q}x$ and $[\begin{smallmatrix} b_0^{(1)} \\ b_o^{(2)} \end{smallmatrix}] = [\begin{smallmatrix} \tilde{Q}^{(1)H} b \\ \tilde{Q}^{(2)H} b \end{smallmatrix}]$ we find that the Moore–Penrose solution satisfies

$$A_o y = b_o^{(2)},$$

which is now a system with linearly independent rows that can be solved exactly with the techniques of [6, 7], leaving an irreducible residue $b_o^{(1)}$. Hence the least squares error will be $\|b_o^{(1)}\|$.

We use the following 4×4-block SSS matrix as an example to illustrate the procedure:

$$A = \begin{bmatrix} D_1 & U_1V_2^H & U_1W_2V_3^H & U_1W_2W_3V_4^H \\ P_2Q_1^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ P_3R_2Q_1^H & P_3Q_2^H & D_3 & U_3V_4^H \\ P_4R_3R_2Q_1^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{bmatrix}. \tag{2}$$

2.1. Right multiplication—Converting to a block upper triangular matrix.

Step 1.

1. Compute $Q_1 = [\,\hat{Q}_{11} \quad \hat{Q}_{12}\,][\begin{smallmatrix} 0 \\ \hat{R}_2 \end{smallmatrix}]$ via QL factorization. $\hat{R}_2$ is a square matrix.
2. Right multiply $[\,\hat{Q}_{11} \quad \hat{Q}_{12}\,]$ to the first block column of A and obtain

$$\begin{bmatrix} D_1\hat{Q}_{11} & D_1\hat{Q}_{12} & U_1V_2^H & U_1W_2V_3^H & U_1W_2W_3V_4^H \\ 0 & P_2\hat{R}_2^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ 0 & P_3R_2\hat{R}_2^H & P_3Q_2^H & D_3 & U_3V_4^H \\ 0 & P_4R_3R_2\hat{R}_2^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{bmatrix}$$

$$= \begin{bmatrix} A_1^{(1)} & \begin{bmatrix} D_1\hat{Q}_{12} & U_1V_2^H \\ P_2\hat{R}_2^H & D_2 \\ P_3R_2\hat{R}_2^H & P_3Q_2^H \\ P_4R_3R_2\hat{R}_2^H & P_4R_3Q_2^H \end{bmatrix} & \tilde{A}_1^{(1)} \end{bmatrix}. \tag{3}$$

Step 2.

1. Compute $\left[\begin{smallmatrix} \hat{R}_2 R_2^H \\ Q_2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} \hat{Q}_{21}^{(1)} & \hat{Q}_{22}^{(1)} \\ \hat{Q}_{21}^{(2)} & \hat{Q}_{22}^{(2)} \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ \hat{R}_3 \end{smallmatrix}\right]$ via QL factorization. $\hat{R}_3$ is a square matrix.
2. Right multiply $\left[\begin{smallmatrix} \hat{Q}_{21}^{(1)} & \hat{Q}_{22}^{(1)} \\ \hat{Q}_{21}^{(2)} & \hat{Q}_{22}^{(2)} \end{smallmatrix}\right]$ to the 2nd and 3rd block columns of A after Step 1, and we obtain

$$\begin{aligned} &\begin{bmatrix} A_1^{(1)} & \begin{bmatrix} [\, D_1\hat{Q}_{12} \quad U_1V_2^H \,]\,\hat{Q}_{21} & [\, D_1\hat{Q}_{12} \quad U_1V_2^H \,]\,\hat{Q}_{22} \\ [\, P_2\hat{R}_2^H \quad D_2 \,]\,\hat{Q}_{21} & [\, P_2\hat{R}_2^H \quad D_2 \,]\,\hat{Q}_{22} \\ 0 & P_3\hat{R}_3^H \\ 0 & P_4R_3\hat{R}_3^H \end{bmatrix} & \tilde{A}_1^{(1)} \end{bmatrix} \\ &= \begin{bmatrix} A_1^{(2)} & \begin{bmatrix} [\, D_1\hat{Q}_{12} \quad U_1V_2^H \,]\,\hat{Q}_{22} & U_1W_2V_3^H \\ [\, P_2\hat{R}_2^H \quad D_2 \,]\,\hat{Q}_{22} & U_2V_3^H \\ P_3\hat{R}_3^H & D_3 \\ P_4R_3\hat{R}_3^H & P_4Q_3^H \end{bmatrix} & \tilde{A}_1^{(2)} \end{bmatrix}. \end{aligned} \tag{4}$$

Step 3.

1. Compute $\left[\begin{smallmatrix} \hat{R}_3 R_3^H \\ Q_3 \end{smallmatrix}\right] = \left[\begin{smallmatrix} \hat{Q}_{31}^{(1)} & \hat{Q}_{32}^{(1)} \\ \hat{Q}_{31}^{(2)} & \hat{Q}_{32}^{(2)} \end{smallmatrix}\right]\left[\begin{smallmatrix} 0 \\ \hat{R}_4 \end{smallmatrix}\right]$ via QL factorization. $\hat{R}_4$ is a square matrix.
2. Right multiply $\left[\begin{smallmatrix} \hat{Q}_{31}^{(1)} & \hat{Q}_{32}^{(1)} \\ \hat{Q}_{31}^{(2)} & \hat{Q}_{32}^{(2)} \end{smallmatrix}\right]$ to the 3rd and 4th block columns of A after Step 2, and we obtain

$$\begin{bmatrix} A_1^{(3)} & \begin{bmatrix} [\,[\, D_1\hat{Q}_{12} \quad U_1V_2^H \,]\,\hat{Q}_{22} \quad U_1W_2V_3^H \,]\,\hat{Q}_{32} & U_1W_2W_3V_4^H \\ [\,[\, P_2\hat{R}_2^H \quad D_2 \,]\,\hat{Q}_{22} \quad U_2V_3^H \,]\,\hat{Q}_{32} & U_2W_3V_4^H \\ [\, P_3\hat{R}_3^H \quad D_3 \,]\,\hat{Q}_{32} & U_3V_4^H \\ P_4\hat{R}_4^H & D_4 \end{bmatrix} \end{bmatrix}, \tag{5}$$

where

$$A_1^{(3)} = \begin{bmatrix} A_1^{(2)} & \begin{bmatrix} [\,[\, D_1\hat{Q}_{12} \quad U_1V_2^H \,]\,\hat{Q}_{22} \quad U_1W_2V_3^H \,]\,\hat{Q}_{31} \\ [\,[\, P_2\hat{R}_2^H \quad D_2 \,]\,\hat{Q}_{22} \quad U_2V_3^H \,]\,\hat{Q}_{31} \\ [\, P_3\hat{R}_3^H \quad D_3 \,]\,\hat{Q}_{31} \\ 0 \end{bmatrix} \end{bmatrix}. \tag{6}$$

2.2. Time-varying system notation representation. We will use the following time-varying system notations to preserve and represent the SSS structure for the reduction procedure described above. If A is a block upper triangular matrix with an SSS representation, we use

$$\langle A \rangle_i = \left[\begin{array}{c|c} W_i & V_i^H \\ \hline U_i & D_i \end{array}\right],$$

collecting the ith components of the SSS representation in a single block 2×2 matrix. In the above notation, some of the block rows or columns may disappear. The dimensions of the blocks have to be congruent, of course. The entry $A_{i,j}$ for $i < j$ is given by $A_{i,j} = U_i W_{i+1} \cdots W_{j-1} V_j$ and $A_{i,i} = D_i$.

We use the terms "realization" and "representation" indiscriminately to indicate the matrices used in diverse semiseparable representations. In the next sections we assume that all the given representations are of minimal dimensions. (They will be if the construction procedures of [6] have been used.)

With these notations, we rewrite the right multiplication procedure as follows.

Step 1.

1. Compute $Q_1 = [\,\hat{Q}_{11} \quad \hat{Q}_{12}\,][\begin{smallmatrix} 0 \\ \hat{R}_2 \end{smallmatrix}]$ via QL.
2. $\langle \hat{Q} \rangle_1$

$$= \left[\begin{array}{c|c} \cdot & \cdot \\ \hline \hat{Q}_{12} & \hat{Q}_{11} \end{array}\right],$$

where "$\cdot$" denotes a 0-dimension matrix.

3. $\langle A_1 \rangle_1$

$$= \left[\begin{array}{c|c} \cdot & \cdot \\ \hline [\,D_1\hat{Q}_{12} \quad U_1\,] & D_1\hat{Q}_{11} \end{array}\right].$$

Step 2.

1. Compute $[\begin{smallmatrix} \hat{R}_2 R_2^H \\ Q_2 \end{smallmatrix}] = [\begin{smallmatrix} \hat{Q}_{21}^{(1)} & \hat{Q}_{22}^{(1)} \\ \hat{Q}_{21}^{(2)} & \hat{Q}_{22}^{(2)} \end{smallmatrix}][\begin{smallmatrix} 0 \\ \hat{R}_3 \end{smallmatrix}]$ via QL.
2. $\langle [\hat{Q} \rangle_2$

$$= \left[\begin{array}{c|c} \hat{Q}_{22}^{(1)} & \hat{Q}_{21}^{(1)} \\ \hline \hat{Q}_{22}^{(2)} & \hat{Q}_{21}^{(2)} \end{array}\right].$$

3. $\langle A_1 \rangle_2$

$$= \left[\begin{array}{c|c} \begin{bmatrix} \hat{Q}_{22}^{(1)} & 0 \\ V_2^H \hat{Q}_{22}^{(2)} & W_2 \end{bmatrix} & \begin{bmatrix} \hat{Q}_{21}^{(1)} \\ V_2^H \hat{Q}_{21}^{(2)} \end{bmatrix} \\ \hline \left[\, P_2 \hat{R}_2^H \hat{Q}_{22}^{(1)} + D_2 \hat{Q}_{22}^{(2)} \quad U_2 \,\right] & P_2 \hat{R}_2^H \hat{Q}_{21}^{(1)} + D_2 \hat{Q}_{21}^{(2)} \end{array}\right].$$

On completing this step, we are able to construct the first two block columns of A_1:

$$A_1 = \begin{bmatrix} D_1\hat{Q}_{11} & [\,D_1\hat{Q}_{12} \quad U_1\,]\begin{bmatrix} \hat{Q}_{21}^{(1)} \\ V_2^H \hat{Q}_{21}^{(2)} \end{bmatrix} & ? & ? \\ 0 & P_2 \hat{R}_2^H \hat{Q}_{21}^{(1)} + D_2 \hat{Q}_{21}^{(2)} & ? & ? \\ 0 & 0 & ? & ? \\ 0 & 0 & ? & ? \end{bmatrix} = \begin{bmatrix} A_1^{(2)} & \begin{bmatrix} ? & ? \\ ? & ? \\ ? & ? \\ ? & ? \end{bmatrix} \end{bmatrix}. \tag{7}$$

Step 3.

1. Compute $[\begin{smallmatrix} \hat{R}_3 R_3^H \\ Q_3 \end{smallmatrix}] = [\begin{smallmatrix} \hat{Q}_{31}^{(1)} & \hat{Q}_{32}^{(1)} \\ \hat{Q}_{31}^{(2)} & \hat{Q}_{32}^{(2)} \end{smallmatrix}][\begin{smallmatrix} 0 \\ \hat{R}_4 \end{smallmatrix}]$ via QL.
2. $\langle \hat{Q} \rangle_3$

$$= \left[\begin{array}{c|c} \hat{Q}_{32}^{(1)} & \hat{Q}_{31}^{(1)} \\ \hline \hat{Q}_{32}^{(2)} & \hat{Q}_{31}^{(2)} \end{array}\right].$$

3. $\langle A_1 \rangle_3$

$$= \left[\begin{array}{c|c} \begin{bmatrix} \hat{Q}_{32}^{(1)} & 0 \\ V_3^H \hat{Q}_{32}^{(2)} & W_3 \end{bmatrix} & \begin{bmatrix} \hat{Q}_{31}^{(1)} \\ V_3^H \hat{Q}_{31}^{(2)} \end{bmatrix} \\ \hline \left[\, P_3 \hat{R}_3^H \hat{Q}_{32}^{(1)} + D_3 \hat{Q}_{32}^{(2)} \quad U_3 \,\right] & P_3 \hat{R}_3^H \hat{Q}_{31}^{(1)} + D_3 \hat{Q}_{31}^{(2)} \end{array}\right].$$

On completing this step, we are able to construct the first three block columns of A_1:

$$A_1 = \left[\begin{array}{cc} A_1^{(2)} & \left[\begin{array}{cc} [\,D_1\hat{Q}_{12} \quad U_1\,]\begin{bmatrix} \hat{Q}_{21}^{(1)} & 0 \\ V_2^H\hat{Q}_{21}^{(2)} & W_2 \end{bmatrix}\begin{bmatrix} \hat{Q}_{32}^{(1)} \\ V_3^H\hat{Q}_{32}^{(2)} \end{bmatrix} & ? \\ \left[\, P_2\hat{R}_2^H\hat{Q}_{22}^{(1)} + D_2\hat{Q}_{22}^{(2)} \quad U_2\,\right]\begin{bmatrix} \hat{Q}_{32}^{(1)} \\ V_3^H\hat{Q}_{32}^{(2)} \end{bmatrix} & ? \\ P_3\hat{R}_3^H\hat{Q}_{31}^{(1)} + D_3\hat{Q}_{31}^{(2)} & ? \\ 0 & ? \end{array}\right] \end{array}\right] = \left[\begin{array}{cc} A_1^{(3)} & \begin{bmatrix} ? \\ ? \\ ? \\ ? \end{bmatrix} \end{array}\right]. \tag{8}$$

Step 4. This is the last step for our 4×4-block example, and there is nothing left to be done except the calculation of the last block column:

$$\langle A_1 \rangle_4 = \left[\begin{array}{c|c} \cdot & \begin{bmatrix} I & 0 \\ 0 & V_4^H \end{bmatrix} \\ \hline \cdot & [\,P_4\hat{R}_4^H \quad D_4\,] \end{array}\right].$$

And, finally, we have the complete block upper triangular matrix A_1:

$$A_1 = \left[\begin{array}{cc} A_1^{(3)} & \left[\begin{array}{c} [\,D_1\hat{Q}_{12} \quad U_1\,]\begin{bmatrix} \hat{Q}_{22}^{(1)} & 0 \\ V_2^H\hat{Q}_{22}^{(2)} & W_2 \end{bmatrix}\begin{bmatrix} \hat{Q}_{32}^{(1)} & 0 \\ V_3^H\hat{Q}_{32}^{(2)} & W_3 \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & V_4^H \end{bmatrix} \\ \left[\, P_2\hat{R}_2^H\hat{Q}_{22}^{(1)} + D_2\hat{Q}_{22}^{(2)} \quad U_2\,\right]\begin{bmatrix} \hat{Q}_{32}^{(1)} & 0 \\ V_3^H\hat{Q}_{32}^{(2)} & W_3 \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & V_4^H \end{bmatrix} \\ \left[\, P_3\hat{R}_3^H\hat{Q}_{32}^{(1)} + D_3\hat{Q}_{32}^{(2)} \quad U_3\,\right]\begin{bmatrix} I & 0 \\ 0 & V_4^H \end{bmatrix} \\ [\,P_4\hat{R}_4^H \quad D_4\,] \end{array}\right] \end{array}\right]. \tag{9}$$

For the simplicity of notation in the following steps, we denote the component matrices as follows:

$$\hat{U}_i = \left[\, P_i\hat{R}_i^H\hat{Q}_{i2}^{(1)} + D_i\hat{Q}_{i2}^{(2)} \quad U_i\,\right],$$

$$\hat{V}_i^H = \begin{bmatrix} \hat{Q}_{i2}^{(1)} \\ V_i^H\hat{Q}_{i2}^{(2)} \end{bmatrix}, \text{ except for } \hat{V}_4^H = \begin{bmatrix} I & 0 \\ 0 & V_4^H \end{bmatrix},$$

$$\hat{W}_i = \begin{bmatrix} \hat{Q}_{i2}^{(1)} & 0 \\ V_i^H\hat{Q}_{i2}^{(2)} & W_i \end{bmatrix},$$

and

$$\hat{D}_i = P_i\hat{R}_i^H\hat{Q}_{i1}^{(1)} + D_i\hat{Q}_{i1}^{(2)}, \text{ except for } \hat{D}_4 = [\,P_4\hat{R}_4^H \quad D_4\,].$$

With these notations,

$$A_1 = A\hat{Q} = \begin{bmatrix} \hat{D}_1 & \hat{U}_1\hat{V}_2^H & \hat{U}_1\hat{W}_2\hat{V}_3^H & \hat{U}_1\hat{W}_2\hat{W}_3\hat{V}_4^H \\ 0 & \hat{D}_2 & \hat{U}_2\hat{V}_3^H & \hat{U}_2\hat{W}_3\hat{V}_4^H \\ 0 & 0 & \hat{D}_3 & \hat{U}_3\hat{V}_4^H \\ 0 & 0 & 0 & \hat{D}_4 \end{bmatrix}.$$

2.3. Left multiplication—Converting to linearly independent rows.

Step 1. QR decomposition on $[\,\hat{D}_1 \quad \hat{U}_1\,]$:

$$\begin{bmatrix} \tilde{Q}_1^{(1)} & \tilde{Q}_1^{(2)} & \tilde{Q}_1^{(3)} \end{bmatrix}^H [\,\hat{D}_1 \quad \hat{U}_1\,] = \begin{bmatrix} 0 & 0 \\ \hat{D}_{o1} & \hat{U}_{o1} \\ 0 & Y_2 \end{bmatrix},$$

where $\tilde{Q}_1 = [\,\tilde{Q}_1^{(1)} \quad \tilde{Q}_1^{(2)} \quad \tilde{Q}_1^{(3)}\,]$ is the "Q" matrix of the QR decomposition (for later convenience and consistency we have put the bottom row of zeros in the "R" factor on top), and $\hat{D}_{o1}$ and Y_2 have linearly independent rows by construction. (They can be in echelon form.) The result of Step 1 applied to the whole matrix A_1 by left multiplying the elementary transformation $\tilde{Q}_1^H$ is

$$\left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ \hat{D}_{o1} & \hat{U}_{o1}\hat{V}_2^H & \hat{U}_{o1}\hat{W}_2\hat{V}_3^H & \hat{U}_{o1}\hat{W}_2\hat{W}_3\hat{V}_4^H \\ \hline 0 & Y_2\hat{V}_2^H & Y_2\hat{W}_2\hat{V}_3^H & Y_2\hat{W}_2\hat{W}_3\hat{V}_4^H \\ \hline\hline 0 & \hat{D}_2 & \hat{U}_2\hat{V}_3^H & \hat{U}_2\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & \hat{D}_3 & \hat{U}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & \hat{D}_4 \end{array}\right],$$

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. This step consists again in finding a QR factorization,

$$\tilde{Q}_2^H \begin{bmatrix} Y_2\hat{V}_2^H & Y_2\hat{W}_2 \\ \hat{D}_2 & \hat{U}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \hat{D}_{o2} & \hat{U}_{o2} \\ 0 & Y_3 \end{bmatrix},$$

in which $\tilde{Q}_2$ has the block decomposition

$$\tilde{Q}_2 = \begin{bmatrix} \tilde{Q}_{21}^{(1)} & \tilde{Q}_{21}^{(2)} & \tilde{Q}_{21}^{(3)} \\ \tilde{Q}_{22}^{(1)} & \tilde{Q}_{22}^{(2)} & \tilde{Q}_{22}^{(3)} \end{bmatrix}.$$

Left multiplying $\tilde{Q}_2^H$ produces a result for the second step:

$$\left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ \hat{D}_{o1} & \hat{U}_{o1}\hat{V}_2^H & \hat{U}_{o1}\hat{W}_2\hat{V}_3^H & \hat{U}_{o1}\hat{W}_2\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & 0 \\ 0 & \hat{D}_{o2} & \hat{U}_{o2}\hat{V}_3^H & \hat{U}_{o2}\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & Y_3\hat{V}_3^H & Y_3\hat{W}_3\hat{V}_4^H \\ \hline\hline 0 & 0 & \hat{D}_3 & \hat{U}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & \hat{D}_4 \end{array}\right].$$

Step 3. Again, we perform a QR factorization,

$$\tilde{Q}_3^H \begin{bmatrix} Y_3\hat{V}_3^H & Y_3\hat{W}_3 \\ \hat{D}_3 & \hat{U}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \hat{D}_{o3} & \hat{U}_{o3} \\ 0 & Y_4 \end{bmatrix},$$

in which $\tilde{Q}_3$ has the block decomposition

$$\tilde{Q}_2 = \begin{bmatrix} \tilde{Q}_{31}^{(1)} & \tilde{Q}_{31}^{(2)} & \tilde{Q}_{31}^{(3)} \\ \tilde{Q}_{32}^{(1)} & \tilde{Q}_{32}^{(2)} & \tilde{Q}_{32}^{(3)} \end{bmatrix}.$$

Left multiplying $\tilde{Q}_3^H$ we obtain

$$\left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ \hat{D}_{o1} & \hat{U}_{o1}\hat{V}_2^H & \hat{U}_{o1}\hat{W}_2\hat{V}_3^H & \hat{U}_{o1}\hat{W}_2\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & 0 \\ 0 & \hat{D}_{o2} & \hat{U}_{o2}\hat{V}_3^H & \hat{U}_{o2}\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & \hat{D}_{o3} & \hat{U}_{o3}\hat{V}_4^H \\ \hline 0 & 0 & 0 & Y_4\hat{V}_4^H \\ \hline\hline 0 & 0 & 0 & \hat{D}_4 \end{array}\right].$$

Step 4. In this final step, we QR factorize $[\begin{smallmatrix} Y_4\hat{V}_4^H \\ \hat{D}_4 \end{smallmatrix}]$:

$$\tilde{Q}_4^H \begin{bmatrix} Y_4\hat{V}_4^H \\ \hat{D}_4 \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{D}_{o4} \end{bmatrix},$$

in which the last block consists of linearly independent rows to yield the final result:

$$\left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ \hat{D}_{o1} & \hat{U}_{o1}\hat{V}_2^H & \hat{U}_{o1}\hat{W}_2\hat{V}_3^H & \hat{U}_{o1}\hat{W}_2\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & 0 \\ 0 & \hat{D}_{o2} & \hat{U}_{o2}\hat{V}_3^H & \hat{U}_{o2}\hat{W}_3\hat{V}_4^H \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & \hat{D}_{o3} & \hat{U}_{o3}\hat{V}_4^H \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \hat{D}_{o4} \end{array}\right].$$

If we left multiply the $\tilde{Q}_i^H$'s to b also, we will have

$$\tilde{Q}_4^H\tilde{Q}_3^H\tilde{Q}_2^H\tilde{Q}_1^H \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \left[\begin{array}{c} b_{o1}^{(1)} \\ b_{o1}^{(2)} \\ \hline b_{o2}^{(1)} \\ b_{o2}^{(2)} \\ \hline b_{o3}^{(1)} \\ b_{o3}^{(2)} \\ \hline b_{o4}^{(1)} \\ b_{o4}^{(2)} \end{array}\right].$$

Hence, the quantities $b_{oi}^{(1)}$'s remain to produce the Moore–Penrose square error:

$$\epsilon_{MPI}^2 = \|b_{o1}^{(1)}\|^2 + \|b_{o3}^{(1)}\|^2 + \|b_{o3}^{(1)}\|^2 + \|b_{o4}^{(1)}\|^2,$$

TABLE 2
The dimensions of the component matrices used in the reduction procedure.

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

Matrix	$\hat{R}_i$	$\hat{Q}_{i1}^{(1)}$	$\hat{Q}_{i1}^{(2)}$	$\hat{Q}_{i2}^{(1)}$	$\hat{Q}_{i2}^{(2)}$
Dimensions	$l_i \times l_i$	$l_i \times (n_i + l_i - l_{i+1})$	$n_i \times (n_i + l_i - l_{i+1})$	$l_i \times l_{i+1}$	$n_i \times l_{i+1}$

while Y_i's will now be uniquely determined by solving a unilateral semiseparable system given by

$$\begin{bmatrix} \hat{D}_{o1} & \hat{U}_{o1}\hat{V}_2^H & \hat{U}_{o1}\hat{W}_2\hat{V}_3^H & \hat{U}_{o1}\hat{W}_2\hat{W}_3\hat{V}_4^H \\ 0 & \hat{D}_{o2} & \hat{U}_{o2}\hat{V}_3^H & \hat{U}_{o2}\hat{W}_3\hat{V}_4^H \\ 0 & 0 & \hat{D}_{o3} & \hat{U}_{o3}\hat{V}_4^H \\ 0 & 0 & 0 & \hat{D}_{o4} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} b_{o1}^{(2)} \\ b_{o2}^{(2)} \\ b_{o3}^{(2)} \\ b_{o4}^{(2)} \end{bmatrix},$$

and this system can now be solved by the direct method of [6].

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 time-varying 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.

2.4. Flop counts. We list the dimensions of each component matrix in Table 2. We have $\sum_{i=1}^{K} m_i = M$ and $\sum_{i=1}^{K} n_i = N$.

In each step of the one-pass, top-down algorithm, the following tasks must be performed, and the flops required for each task are listed:

1. Multiply a triangular matrix to a general matrix $\hat{R}_i R_i^H$: $l_i^2 l_{i+1}$ flops required.
2. QR factorize a general matrix $[\begin{smallmatrix} \hat{R}_i R_i^H \\ Q_i \end{smallmatrix}]$ via Householder triangularization: $2l_{i+1}^2(l_i + n_i) - \frac{2}{3}l_{i+1}^3$ flops required.
3. Left multiply a triangular matrix to a general matrix $P_i \hat{R}_i$: $m_i l_i^2$ flops.
4. Right multiply $\hat{Q}_i$ generated in the above step to
$$\begin{bmatrix} I & 0 \\ 0 & V_i^H \\ P_i & D_i \end{bmatrix},$$
using the elementary reflectors: $[4(l_i + n_i)l_{i+1} - 2l_{i+1}^2](l_i + k_i + m_i)$ flops.
5. Left multiply a triangular matrix to a general matrix $Y_i[\,\hat{V}_i^H \quad \hat{W}_i\,]$: $(l_i + k_i)^2(l_i + n_i + k_{i+1})$ flops.
6. QR factorize a general matrix $[\begin{smallmatrix} Y_i\hat{V}_i^H & \hat{W}_i \\ \hat{D}_i & \hat{U}_i \end{smallmatrix}]$ via Householder triangularization: $2(l_i + n_i + k_{i+1})^2(l_i + k_i + m_i) - \frac{2}{3}(l_i + n_i + k_{i+1})^3$ flops.

Assuming $m_i = m = M/K$, $n_i = n/K$, and $l_i = k_i = p$ for all $i = 1, 2, \ldots, K$,[1] we have the approximate total flops

$$(10) \qquad 25p^3K + 22p^2N + 11p^2M + 12pM/K + 2N^2M/K - \frac{2}{3}N^2/K^2.$$

We now approximately estimate an optimal number of blocks K, which gives an upper bound of the minimal flops needed, under two conditions:

[1]There is an underlying assumption that M and N have a common divider K with moderate size.

1. $M \approx N$. (10) is approximated by

$$f_1(K) = 25p^3K + 33p^2M + 12pM^2/K + \frac{4}{3}M^3/K^2.$$

Take the derivative with respect to K and set the derivative to 0:

$$25p^3 - 12pM^2/K^2 - \frac{8}{3}M^3/K^3 = 0.$$

$f_1^{\text{opt}} < (33 + \frac{25}{9} + 12\sqrt{3})p^2M \approx 56.56p^2M$, with $K^{\text{opt}} \approx \frac{2\sqrt{3}M}{5p} \approx 0.6928M/p$.

2. $\alpha = \frac{N}{M} \ll 1$. Then (10) is approximated by

$$f_2(K) = 25p^3K + 11p^2M + 12\alpha pM^2/K + 2\alpha^2M^3/K^2.$$

$f_2^{\text{opt}} < (11 + 12\sqrt[3]{25\alpha})p^2M$, with $K^{\text{opt}} \approx \sqrt[3]{\frac{\alpha^2}{25}}\frac{M}{p} \approx 0.3420\sqrt[3]{\alpha^2}M/p$.

2.5. Experimental run-times. We have written a Fortran 90 program to implement our fast solver for the least squares problem. Our experiments were run on a Sun Fire v880 Server with four 900 MHz UltraSPARC-III processors, 8 gigabytes of memory, and the Solaris 2.8 operating system. The code was compiled using Sun WorkShop Fortran 95 compiler and the Sun Performance Library, with options "-fast -dalign." The machine precision is $\epsilon = 2^{-54} \approx 1.1 \times 10^{-16}$. We compared our method with the LAPACK routine DGELS [3], which solves the problem as a general least squares problem.

We used the results in [26] to measure the backward errors in the solutions produced by both methods. Let $\widehat{x}$ be any approximate solution to the least squares problem; and let $A = Q\binom{D}{0}W^T$ be the SVD of A. Rewrite

$$b = Q\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad \text{and} \quad r = b - A\,\widehat{x} = Q\begin{pmatrix} r_1 \\ r_2 \end{pmatrix}.$$

We compute the backward error in $\widehat{x}$ as

$$\mathcal{E} \equiv \begin{cases} \dfrac{\|D\,r_1\|_2}{\|r\|_2} & \text{if } \widehat{x} = 0, \\ \min(\eta, \tilde{\sigma}) & \text{otherwise}, \end{cases}$$

where $\eta = \frac{\|r\|_2}{\|\widehat{x}\|_2}$, $\gamma = \|A\,\widehat{x} - b\|_2$, and

$$\tilde{\sigma} = \sqrt{\frac{r_1^T\,D^2\,\left(D^2 + \eta^2 I\right)^{-1}\,r_1}{\gamma^2/\eta^2 + \eta^2\,r_1^T\,\left(D^2 + \eta^2 I\right)^{-2}\,r_1}}.$$

It is known that this backward error is within a factor of 2 from the smallest possible backward error [26].

In our experiments, we let the number of blocks increase but keep the block sizes and off-diagonal block ranks constant. We tested the following types of matrices:

- *Type* I: Well-conditioned random matrices with $m = 30$, $n = 20$, and $k = l = 5$.

- *Type* II: Well-conditioned random matrices with $m = 30$, $n = 20$, and $k = l = 10$.
- *Type* III: Ill-conditioned random matrices with $m = 30$, $n = 20$, and $k = l = 5$.
- *Type* IV: Ill-conditioned random matrices with $m = 30$, $n = 20$, and $k = l = 10$.

Table 3 summarizes our numerical experiments. Our scheme (**NEW**) was consistently and significantly faster than LAPACK (**DGELS**), sometimes by factors exceeding 5000. On the other hand, it produced backward errors that are as small as those of **DGELS**.

3. Superfast solution of $AX = B$.

We now consider the fast 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 fast multiplication algorithm for matrices in SSS form. Such an approach has already been described in [16]. Here we show how to extend the implicit decomposition ideas [6] 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 nonsquare 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 $\{D_i(B)\}_{i=1}^n$, $\{U_i(B)\}_{i=1}^n$, $\{W_i(B)\}_{i=1}^r$, $\{V_i(B)\}_{i=1}^r$, $\{P_i(B)\}_{i=1}^n$, $\{R_i(B)\}_{i=m}^n$, and $\{Q_i(B)\}_{i=m}^n$. 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 the 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.

This fast solver is essentially of the same form as the fast solver presented in [6]. This fast solver can also be presented in a recursive fashion, which we proceed to do.

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

First we perform orthogonal eliminations by computing QL and LQ factorizations

$$U_1(A) = q_1 \begin{pmatrix} 0 \\ \hat{U}_1(A) \end{pmatrix} \begin{matrix} m_1 - k_1(A) \\ k_1(A) \end{matrix}$$

Table 3
Execution times and scaled backward errors.

		Number of blocks				
		10	20	40	80	160
Execution time (seconds)						
NEW	TYPE I	1.0×10^{-2}	1.6×10^{-2}	2.7×10^{-2}	5.3×10^{-2}	1.2×10^{-1}
	TYPE II	1.5×10^{-2}	2.3×10^{-2}	4.5×10^{-2}	9.9×10^{-2}	2.2×10^{-1}
	TYPE III	1.1×10^{-2}	1.7×10^{-2}	3.2×10^{-2}	5.3×10^{-2}	9.4×10^{-2}
	TYPE IV	1.4×10^{-2}	2.4×10^{-2}	4.4×10^{-2}	8.6×10^{-2}	1.8×10^{-1}
DGELS	TYPE I	3.8×10^{-2}	4.8×10^{-1}	2.3×10^{0}	7.2×10^{1}	5.7×10^{2}
	TYPE II	5.3×10^{-2}	3.3×10^{-1}	2.4×10^{0}	5.2×10^{1}	6.2×10^{2}
	TYPE III	3.9×10^{-2}	2.5×10^{-1}	2.5×10^{0}	4.4×10^{1}	4.9×10^{2}
	TYPE IV	3.8×10^{-2}	2.8×10^{-1}	2.1×10^{0}	3.9×10^{1}	5.2×10^{2}
Scaled backward errors $\frac{\mathcal{E}}{\|A\|_2\,\epsilon}$						
NEW	TYPE I	4.0×10^{-2}	4.3×10^{-2}	5.4×10^{-2}	2.8×10^{-2}	1.6×10^{-2}
	TYPE II	3.6×10^{-2}	5.1×10^{-2}	4.1×10^{-2}	2.9×10^{-2}	3.1×10^{-2}
	TYPE III	2.4×10^{-2}	5.4×10^{-2}	2.4×10^{-2}	1.2×10^{-2}	2.4×10^{-2}
	TYPE IV	5.5×10^{-2}	4.8×10^{-2}	3.3×10^{-2}	3.1×10^{-2}	1.7×10^{-2}
DGELS	TYPE I	5.3×10^{-2}	3.4×10^{-2}	2.7×10^{-2}	1.7×10^{-2}	1.3×10^{-2}
	TYPE II	5.3×10^{-2}	3.3×10^{-2}	2.9×10^{-2}	2.6×10^{-2}	1.8×10^{-2}
	TYPE III	3.9×10^{-2}	2.2×10^{-2}	2.3×10^{-2}	1.7×10^{-2}	1.2×10^{-2}
	TYPE IV	5.1×10^{-2}	3.3×10^{-2}	2.3×10^{-2}	2.0×10^{-2}	1.3×10^{-2}

and

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

where q_1 and w_1 are unitary matrices. We now need to apply q_1^H to the first m_1 rows of (11). 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:

$$q_1^H P_1(B) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{P}_1(B) \\ \hat{P}_1(B) \end{pmatrix},$$

$$q_1^H D_1(B) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{D}_1(B) \\ \tilde{D}_2(B) \end{pmatrix},$$

$$q_1^H U_1(B) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{U}_1(B) \\ \hat{U}_1(B) \end{pmatrix}.$$

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:

$$w_1 P_1(X) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{P}_1(X) & 0 \\ \hat{P}_{11}(X) & \hat{P}_{12}(X) \end{pmatrix},$$

$$w_1 D_1(X) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{D}_1(X) \\ \hat{D}_1(X) \end{pmatrix},$$

$$w_1 U_1(X) = \begin{array}{l} m_1 - k_1(A) \\ k_1(A) \end{array} \begin{pmatrix} \tilde{U}_1(X) & 0 \\ \hat{U}_{11}(X) & \hat{U}_{12}(X) \end{pmatrix}.$$

The zeros in the modified $w_1 U_1(X)$ and $w P_1(X)$ are not a restriction since the number of columns in either of these matrices has not been fixed yet.

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 need only 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

$$D_{11}(A)\begin{pmatrix} Q_m^H(X)R_{m+1}^H(X)\cdots R_0^H(X)\left(\tilde{P}_1(X) \quad 0\right)^H \\ \vdots \\ Q_0^H(X)\left(\tilde{P}_1(X) \quad 0\right)^H \\ \tilde{D}_1^H(X) \\ V_2(X)\left(\tilde{U}_1(X) \quad 0\right)^H \\ \vdots \\ V_r(X)W_{r-1}^H(X)\cdots W_2^H(X)\left(\tilde{U}_1(X) \quad 0\right)^H \end{pmatrix}^H$$

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

Solving this we obtain

$$\begin{aligned} \tilde{D}_1(X) &= D_{11}^{-1}(A)\tilde{D}_1(B), \\ \tilde{P}_1(X) &= D_{11}^{-1}(A)\tilde{P}_1(B), \\ Q_i(X) &= \begin{pmatrix} Q_i(B) & \vec{Q}_i \end{pmatrix}, \\ R_i(X) &= \begin{pmatrix} R_i(B) & 0 \\ (\vec{R}_i)_1 & (\vec{R}_i)_2 \end{pmatrix}, \\ \tilde{U}_1(X) &= D_{11}^{-1}(A)\tilde{U}_1(B), \\ V_i(X) &= \begin{pmatrix} V_i(B) & \vec{V}_i \end{pmatrix}, \\ W_i(X) &= \begin{pmatrix} W_i(B) & 0 \\ (\vec{W}_i)_1 & (\vec{W}_i)_2 \end{pmatrix}. \end{aligned}$$

Notice that all $Q_i(X)$, $V_i(X)$, $R_i(X)$, and $W_i(X)$ have constrained forms for their solutions for all i. These forms will be maintained consistently throughout the algorithm.

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 unknowns from the corresponding rows of B. We observe that it leads to the following changes:

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

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:

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

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 their corresponding hatted forms. The resulting hatted system $\hat{A}\hat{X} = \hat{B} - (\tau \text{ terms})$ looks exactly like the (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

$$\begin{aligned}
D_1(X) &= w_1^H \begin{pmatrix} \tilde{D}_1(X) \\ \hat{D}_1(X) \end{pmatrix}, \\
\hat{U}_1(X) &= (\, \hat{U}_{11}(X) \quad \hat{U}_{12}(X) \,), \\
U_1(X) &= w_1^H \begin{pmatrix} \tilde{U}_1(X) & 0 \\ \hat{U}_{11}(X) & \hat{U}_{12}(X) \end{pmatrix}, \\
\hat{P}_1(X) &= (\, \hat{P}_{11}(X) \quad \hat{P}_{12}(X) \,), \\
P_1(X) &= w_1^H \begin{pmatrix} \tilde{P}_1(X) & 0 \\ \hat{P}_{11}(X) & \hat{P}_{12}(X) \end{pmatrix}.
\end{aligned}$$

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

3.2. Case for $n > 1$ and $k_1(A) \geq m_1$: Merge. The second possibility is that $k_1(A)$ is not large enough to permit efficient elimination. In this case we merge 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. To merge the first two block rows and columns of A, we can use the formulas in section 3.6.

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

$$\begin{aligned}
\tilde{D}_1(B) &= \begin{pmatrix} U_1(B)V_2^H(B) \\ D_2(B) \end{pmatrix}, \\
\hat{D}_i(B) &= D_{i+1}(B), \quad i = 2, \ldots, n-1, \\
\begin{pmatrix} \tilde{P}_1(B) \\ \hat{R}_1(B) \\ \tilde{P}_1(\tau) \end{pmatrix} (\, \hat{R}_0(B) \quad \hat{Q}_0^H(B) \,) &= \begin{pmatrix} \begin{pmatrix} P_1(B) \\ P_2(B)R_1(B) \end{pmatrix} & \begin{pmatrix} D_1(B) \\ P_2(B)Q_1^H(B) \end{pmatrix} \\ R_2(B)R_1(B) & R_2(B)Q_1^H(B) \\ P_1(\tau) & D_1(\tau) \end{pmatrix}, \\
\hat{P}_i(B) &= P_{i+1}(B), \quad n = 2, \ldots, n-1, \\
\hat{R}_i(B) &= R_{i+1}(B), \quad i = m-1, \ldots, -1, 2, \ldots, r-1, \\
\hat{Q}_i(B) &= Q_{i+1}(B), \quad n = m-1, \ldots, -1, 1, \ldots, r-1, \\
\tilde{U}_1(B) &= \begin{pmatrix} U_1(B)W_2(B) \\ U_2(B) \end{pmatrix}, \\
\hat{U}_i(B) &= U_{i+1}(B), \quad i = 2, \ldots, n-1, \\
\hat{W}_i(B) &= W_{i+1}(B), \quad i = 2, \ldots, r-1, \\
\hat{V}_i(B) &= V_{i+1}(B), \quad i = 2, \ldots, r-1.
\end{aligned}$$

The implied factorization in the above formulas must be carried out as follows:

$$\tilde{P}_1(B) = \begin{pmatrix} 0 & I & 0 \\ P_2(B) & 0 & 0 \end{pmatrix},$$

$$\begin{aligned}
\hat{R}_1(B) &= \begin{pmatrix} R_2(B) & 0 & 0 \end{pmatrix}, \\
\tilde{P}_1(\tau) &= \begin{pmatrix} 0 & 0 & I \end{pmatrix}, \\
\hat{R}_0(B) &= \begin{pmatrix} R_1(B) \\ P_1(B) \\ P_1(\tau) \end{pmatrix}, \\
\hat{Q}_0(B) &= \begin{pmatrix} Q_1(B) & D_1^H(B) & D_1^H(\tau) \end{pmatrix}.
\end{aligned}$$

The formal expressions for X are similar to B, so we skip them.

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. Observe that $\tilde{P}_1(\tau)$ has already been defined. The remaining changes are

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

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

$$D_1(X) = \tilde{P}_1(X)\hat{Q}_0^H(X),$$

$$\begin{aligned}
P_1(X) &= \tilde{P}_1(X)\hat{R}_0(X), \\
P_i(X) &= \hat{P}_{i-1}(X), \quad i = 3, \dots, n, \\
R_i(X) &= \hat{R}_{i-1}(X), \quad i = m, \dots, n, \\
Q_i(X) &= \hat{Q}_{i-1}(X), \quad i = m, \dots, n, \\
U_1(X) \begin{pmatrix} W_2(X) & V_2^H(X) \end{pmatrix} &= \begin{pmatrix} \tilde{U}_1(X) & \tilde{D}_1(X) \end{pmatrix}, \\
W_i(X) &= \hat{W}_{i-1}(X), \quad i = 3, \dots, n, \\
V_i(X) &= \hat{V}_{i-1}(X), \quad i = 3, \dots, n,
\end{aligned}$$

where the implied factorization must be solved as follows:

$$\begin{aligned}
U_1(X) &= \begin{pmatrix} 0 & I \end{pmatrix}, \\
\tilde{U}_1(X) &= \begin{pmatrix} \tilde{U}_{11}(X) & \tilde{U}_{12}(X) \end{pmatrix}, \\
W_2(X) &= \begin{pmatrix} W_2(B) & 0 \\ \tilde{U}_{11}(X) & \tilde{U}_{12}(X) \end{pmatrix}, \\
V_2(X) &= \begin{pmatrix} V_2(B) & \tilde{D}_1^H(X) \end{pmatrix}.
\end{aligned}$$

3.3. 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.

3.4. Experimental run-times. We now report on the CPU run-times of the superfast solver on a PowerBook 400 MHz G4 computer with 1 MB of L2 cache, 1 GB of RAM, and a 100 MHz bus. The CPU run-times in seconds are reported in Table 4. We assume that we are solving systems of the form $AX = B$, with B a square matrix. Tests were done on systems where the number of columns of B ranged from 256 to 32,768. This is the size column in Table 4. For the matrix A we chose SSS representations such that $m_i(A) = l_i(A) = k_i(A)$, and $m_i(A)$ was either 16 or 32. For the SSS representation of B we chose $l_i(B) = k_i(B)$, and $k_i(B)$ was either 1 or 4. Of course we chose $m_i(B) = m_i(A)$.

For comparison we also report in Table 4 the CPU run-times of the fast solver of [6] for the same system of equations. Of course the fast solver does not exploit the SSS representation of the right-hand side matrix B, so it is just passed a standard dense matrix representation.

As can be seen the fast solver behaves like an $O(n^2)$ algorithm, where n is the order of the system. While this is an order of magnitude better than the standard Gaussian elimination solvers, it is an order of magnitude slower than the superfast solver. We also observe that the superfast solver is reasonably insensitive to the ranks of the off-diagonal blocks of B.

Table 4

CPU run-times in seconds for the superfast solver. Numbers prefixed with an asterisk denote estimated run-times.

		Size							
$m_i(A)/l_i(A)/k_i(A)$	$k_i(B)/l_i(B)$	256	512	1024	2048	4096	8192	16,384	32,768
16	1	0.16	0.30	0.60	1.24	2.52	5.52	11.10	24.22
16	4	0.17	0.32	0.64	1.30	2.69	5.57	11.68	25.42
32	1	0.35	0.72	1.56	3.27	6.74	14.13	30.21	69.97
32	4	0.35	0.75	1.63	3.33	6.93	14.41	30.99	70.88
16	fast solver	0.33	1.92	7.87	31.19	136.24	544.46	*2.2E3	*8.7E3
32	fast solver	0.49	2.39	9.79	38.65	163.36	659.58	*2.6E3	*1.1E4

Table 5

One-norm normalized residual errors $\|AX-B\|_1/(\epsilon_{mach}(\|A\|_1\|X\|_1+\|B\|_1))$ of the superfast solver in double precision for stable SSS representations $\|W_i(A)\|_1 = \|R_i(A)\|_1 = 1 = \|W_i(B)\|_1 = \|R_i(B)\|_1$. Entries much larger than one indicate potential lack of stability.

		Size				
$m_i(A)/l_i(A)/k_i(A)$	$k_i(B)/l_i(B)$	256	512	1024	2048	4096
16	1	1.67	1.07	1.56	3.67	3.43
16	4	0.87	3.66	4.10	3.49	3.01
32	1	1.10	1.04	7.56	6.94	3.82
32	4	1.78	2.96	5.30	9.27	7.83
64	1	0.64	1.21	1.59	5.43	9.25
64	4	0.60	0.89	2.11	3.80	4.30
128	1	0.36	0.58	1.12	1.65	5.32
128	4	0.36	0.60	2.06	2.00	9.45

3.5. Stability. An unstable superfast solver is not very useful. However, even the standard slow solvers are not technically backward stable when there are multiple right-hand sides. For that matter, there is no proof that even ordinary matrix-matrix multiplication is backward stable [23]. Numerical analysts generally attribute this phenomenon to an insufficient number of degrees of freedom in the problem for attaching the backward error when there is more than one right-hand side. Hence there is no simple way to measure how stable such solvers are. What we have found useful is to use the standard normalized measure of backward error [23] for the single right-hand side case, with a straightforward modification:

$$\frac{\|AX-B\|_1}{\epsilon_{mach}(\|A\|_1\|X\|_1+\|B\|_1)}.$$

This is not equivalent to measuring the backward error column by column. In fact it is weaker. The justification is that it can be viewed as a normalized residual.

In Table 5 we present the normalized residual error for a series of random experiments where we used the superfast solver to solve systems of the form $AX = B$, where both A and B were chosen to be random stable SSS forms. As can be seen from the table, the superfast solver seems to behave well from a numerical point of view. However, a more thorough error analysis is needed. This is outside the scope of this paper, so it will be reported separately.

3.6. Merging and splitting SSS blocks. In the presentation of the superfast solver for $AX = B$, we assumed that A and B had SSS representations that were conformally partitioned. If that is not the case, the situation can be remedied by suitable merging and/or splitting partitions in A and B. For that purpose we present some simple formulas in this section.

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:

$$A = \begin{pmatrix} D_1 & U_1V_2^H & U_1W_2V_3^H & U_1W_2W_3V_4^H \\ P_2Q_1^H & D_2 & U_2V_3^H & U_2W_3V_4^H \\ P_3R_2Q_1^H & P_3Q_2^H & D_3 & U_3V_4^H \\ P_4R_3R_2Q_1^H & P_4R_3Q_2^H & P_4Q_3^H & D_4 \end{pmatrix}$$

$$= \begin{pmatrix} D_1 & U_1\begin{pmatrix} V_2^H & W_2V_3^H \end{pmatrix} & U_1\left(W_2W_3\right)V_4^H \\ \begin{pmatrix} P_2 \\ P_3R_2 \end{pmatrix} Q_1^H & \begin{pmatrix} D_2 & U_2V_3^H \\ P_3Q_2^H & D_3 \end{pmatrix} & \begin{pmatrix} U_2W_3 \\ U_3 \end{pmatrix} V_4^H \\ P_4\left(R_3R_2\right)Q_1^H & P_4\begin{pmatrix} R_3Q_2^H & Q_3^H \end{pmatrix} & D_4 \end{pmatrix}.$$

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 ith block:

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

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:

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

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

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

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

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

By factorizing the left-hand side matrices using numerical tools such as the SVD and rank-revealing QR factorizations [13, 27], these two equations allow us to compute an effective representation of those remaining matrices for the new blocks.

3.7. Fast model reduction. As became evident in the presentation of the superfast solver, sometimes we produce SSS representations that are not as compact as possible (to some specified tolerance τ). Dewilde and van der Veen [16] 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 prespecified tolerance τ. This method is closely related to the algorithms presented in [16] for computing minimal realizations in input and output normal forms.

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 speedup for a small loss in compression.

Our algorithm for model reduction can be viewed as a natural extension of the algorithm presented in [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 [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 P_i, R_i, and 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,[2] 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 [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 the left proper form can be restored efficiently. For convenience, we use hats to denote the representation in left proper form.

Consider the following recursions:

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

[2]The term Hankel-block is taken from [16]. It denotes the off-diagonal blocks that extend from the diagonal to the northeast corner (for the upper case) or to the southwest corner (for the lower case).

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

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

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 *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 unhatted 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

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

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 $\tilde{W}$ and then to $\hat{W}_i$.

4. Related and future work.

The algebraic formalism of the structure that is used in this paper appears in work of Dewilde and van der Veen [16] and Eidelman and Gohberg [18, 19, 20, 21]. Other structures that would work just as well appear in the works of Starr [38], Starr and Rokhlin [39], Hackbusch [29], Hackbusch and Khoromskij [30, 31], Hackbusch, Khoromskij, and Sauter [32], and Chandrasekaran and Gu [9]. Fast, direct, but not necessarily stable algorithms for such matrices were first presented in Starr's thesis [38, 39]. They also appear in the works of Eidelman and Gohberg [18, 19] and Hackbusch [29] and Hackbusch and Khoromskij [30, 31]. Fast and stable direct solvers appeared first in the works of Dewilde and van der Veen [16] and independently later in the work of Chandrasekaran and Gu [8]. Various generalizations have been carried out by various authors, including us. Here is a brief list of such works: [10, 11, 12, 30, 31, 32, 34, 36].

The computational electromagnetics literature has also looked at this problem independently. Relevant work includes that of Gurel and Chew [28], Canning and Rogovin [4], and Gope and Jandhyala [24].

In this paper we presented a fast and stable direct method for solving a linear least squares problem whose coefficient matrix satisfies the SSS matrix structure as defined in (1). We also presented a fast solver for handling multiple right-hand sides that are also—in addition to the coefficient matrix—in SSS form.

The fast multipole method (FMM) of Carrier, Greengard, and Rokhlin [5], Greengard and Rokhlin [25], and Rokhlin [37] 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 SSS 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 SSS algorithms to speed up Kress's global discretization technique for solving the integral equations of scattering theory [14, 15].

Acknowledgments. The authors are grateful to Nathan Champagne and Robert Sharpe of Lawrence Livermore National Laboratory for their suggestions and interest in this research and to Jianlin Xia and Jiang Zhu of the University of California at Berkeley for their help in preparing portions of this paper.

REFERENCES

[1] D. Alpay and P. Dewilde, *Time-varying signal approximation and estimation*, in Signal Processing, Scattering, and Operation Theory, and Numerical Methods, Birkhäuser Boston, Boston, MA, 1990, pp. 1–22.

[2] D. Alpay, P. Dewilde, and H. Dym, *Lossless inverse scattering and reproducing kernels for upper triangular matrices*, in Extension and Interpolation of Linear Operators and Matrix Functions, Birkhäuser, Basel, 1990, pp. 61–135.

[3] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, 2nd ed., SIAM, Philadelphia, 1994.

[4] F. X. Canning and K. Rogovin, *Fast direct solution of moment-method matrices*, IEEE Antennas and Propagation Magazine, 40 (1998), pp. 15–26.

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

[6] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen, *Fast stable solver for sequentially semi-separable linear systems of equations*, in High Performance Computing—HiPC 2002: 9th International Conference, Lecture Notes in Comput. Sci. 2552, Springer-Verlag, Heidelberg, 2002, pp. 545–554.

[7] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Fast Stable Solvers for Sequentially Semi-separable Linear Systems of Equations and Least Squares Problems*, Technical report, University of California, Berkeley, CA, 2003.

[8] S. Chandrasekaran and M. Gu, *Fast and stable algorithms for banded plus semiseparable systems of linear equations*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 373–384.

[9] 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, V. Olshevsky, ed., Contemp. Math. 281, AMS, Providence, RI, 2001, pp. 39–53.

[10] S. Chandrasekaran and M. Gu, *Fast and stable eigendecomposition of symmetric banded plus semi-separable matrices*, Linear Algebra Appl., 313 (2000), pp. 107–114.

[11] S. Chandrasekaran and M. Gu, *A divide-and-conquer algorithm for the eigendecomposition of symmetric block-diagonal plus semiseparable matrices*, Numer. Math., 96 (2004), pp. 723–731.

[12] 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.

[13] S. Chandrasekaran and I. C. F. Ipsen, *On rank-revealing QR factorisations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 592–622.

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

[15] D. Colton and R. Kress, *Inverse Acoustic and Electromagnetic Scattering Theory,* Appl. Math. Sci. 93, Springer-Verlag, Berlin, 1992.

[16] P. Dewilde and A. van der Veen, *Time-Varying Systems and Computations*, Kluwer Academic Publishers, Boston, MA, 1998.

[17] P. Dewilde and A. J. van der Veen, *Inner-outer factorization and the inversion of locally finite systems of equations*, Linear Algebra Appl., 313 (2000), pp. 53–100.

[18] Y. Eidelman and I. Gohberg, *Inversion formulas and linear complexity algorithm for diagonal plus semiseparable matrices*, Comput. Math. Appl., 33 (1997), Elsevier, pp. 69–79.

[19] Y. Eidelman and I. Gohberg, *A look-ahead block Schur algorithm for diagonal plus semiseparable matrices*, Comput. Math. Appl., 35 (1998), pp. 25–34.

[20] Y. Eidelman and I. Gohberg, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999), pp. 293–324.

[21] Y. Eidelman and I. Gohberg, *A modification of the Dewilde van der Veen method for inversion of finite structured matrices*, Linear Algebra Appl., 343/344 (2002), pp. 419–450.

[22] I. Gohberg, T. Kailath, and I. Koltracht, *Linear complexity algorithms for semiseparable matrices*, Integral Equations Operator Theory, 8 (1985), pp. 780–804.

[23] G. Golub and C. van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[24] D. Gope and V. Jandhyala, *An iteration-free fast multilevel solver for dense method of moment systems*, in Proceedings of the IEEE meeting on Electrical Performance of Electronic Packaging, 2001, Boston, MA, pp. 177–180.

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

[26] M. Gu, *Backward perturbation bounds for linear least squares problems*, SIAM J. Matrix Anal. Appl., 20 (1998), pp. 363–372.

[27] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.

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

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

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

[31] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic: General complexity estimates*, J. Comput. Appl. Math., 125 (2000), pp. 79–501.

[32] W. Hackbusch, B. N. Khoromskij, and S. Sauter, *On H^2-Matrices*, preprint 50, MPI, Leipzig, 1999.

[33] T. Kailath, *Fredholm resolvents, Wiener-Hopf equations, and Riccati differential equations*, IEEE Trans. Inform. Theory, IT-15 (1969), pp. 655–672.

[34] N. Mastronardi, S. Chandrasekaran, and S. van Huffel, *Fast and stable two-way chasing algorithm for diagonal plus semi-separable systems of linear equations*, Numer. Linear Algebra Appl., 38 (2000), pp. 7–12.

[35] A. P. Mullhaupt and K. S. Riedel, *Low grade matrices and matrix fraction representations*, Linear Algebra Appl., 342 (2002), pp. 187–201.

[36] 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, 41 (2001), pp. 149–157.

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

[38] P. Starr, *On the Numerical Solution of One-Dimensional Integral and Differential Equations*, Research report YALEU/DCS/RR-888, Yale University, New Haven, CT, 1991.

[39] P. Starr and V. Rokhlin, *On the numerical solution of* 2*-point boundary value problem.* II., Comm. Pure Appl. Math., 47 (1994), pp. 1117–1159.

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

[41] N. Yarvin and V. Rokhlin, *A generalized one-dimensional fast multipole method with application to filtering of spherical harmonics*, J. Comput. Phys., 147 (1998), pp. 594–609.