SHIFTED KRONECKER PRODUCT SYSTEMS [*]

CARLA D. MORAVITZ MARTIN [†] AND CHARLES F. VAN LOAN [‡]

Abstract. A fast method for solving a linear system of the form $(A^{(p)} \otimes \cdots \otimes A^{(1)} - \lambda I)x = b$ is given where each $A^{(i)}$ is an n_i-by-n_i matrix. The first step is to convert the problem to triangular form $(T^{(p)} \otimes \cdots \otimes T^{(1)} - \lambda I)y = c$ by computing the (complex) Schur decompositions of the $A^{(i)}$. This is followed by a recursive back-substitution process that fully exploits the Kronecker structure and requires just $O(N(n_1 + \cdots + n_p))$ flops where $N = n_1 \cdots n_p$. A similar method is employed when the real Schur Decomposition is used to convert each $A^{(i)}$ to quasi-triangular form. The numerical properties of these new methods are the same as if we explicitly formed $(T^{(p)} \otimes \cdots \otimes T^{(1)} - \lambda I)$ and used conventional back-substitution to solve for y.

Key words. Linear Systems, Schur decomposition, Back-substitution, Kronecker products

AMS subject classifications. 15A06, 65F05, 65G50

1. Introduction. Matrix problems with replicated block structure abound in signal and image processing, semidefinite programming, control theory, and many other application areas. In these venues fast algorithms have emerged that exploit the rich algebra of the Kronecker product. Perhaps the best example of this is the fast Fourier transform which can be described using the "language" of sparse, matrix factorizations and the Kronecker product. This operation is surfacing more and more as cheap memory prompts the assembly of huge, multidimensional datasets. When techniques for problems of low dimension are generalized or "tensored" together to address a high-dimensional, multilinear problem, then one typically finds a computational challenge that involves the Kronecker product.

It is in the spirit of bringing the fruits of numerical linear algebra to the realm of numerical multilinear algebra that we present the current paper. Our goal is to present a methodology for solving a shifted linear system when the matrix of coefficients is a Kronecker product. Specifically, the question we address is how to solve a shifted Kronecker product system of the form

$$(1.1) \qquad \left(A^{(p)} \otimes \cdots \otimes A^{(1)} - \lambda I_N \right) x = b \qquad b \in \mathbb{R}^N$$

where $A^{(i)} \in \mathbb{R}^{n_i \times n_i}$, $i = 1{:}p$ are given and $N = n_1 \cdots n_p$. A reshaped special case of this problem is the discrete-time Sylvester equation $A^{(1)} X A^{(2)^T} - X = B$. As with many matrix equations of this variety, the first step is to convert $A^{(1)}$ and $A^{(2)}$ to triangular form via the Schur decompoistion. The resulting system can then be solved via a back-substitution process. Jonsson and Kågström [2] have developed block recursive methods for these kinds of problems and they are very effective in high-performance computing environments. The method we present is also recursive and can be regarded as a generalization of their technique. However, we do not generate the subproblems by splitting at the block level.

There are other, well known settings where linear equation solving via the Schur, real Schur, or Hessenberg decompositions is preferred over Gaussian elimination and

[†]Mathematics, Cornell University, 227 Malott Hall, Ithaca, NY 14853-7510, carlam@cam.cornell.edu

[‡]Computer Science, Cornell University, 4130 Upson Hall, Ithaca, NY 14853-7510, cv@cs.cornell.edu

the LU factorization. For example, suppose $A \in \mathbb{R}^{N \times N}$, $b \in \mathbb{R}^N$, $d \in \mathbb{R}^N$ and that we want to explore the behavior of the function

$$f(\lambda) = d^T (A - \lambda I_N)^{-1} b$$

where λ is a scalar. Note that for each λ we must solve a system of linear equations

$$(1.2) \qquad\qquad\qquad (A - \lambda I_N) x = b.$$

If one proceeds to use Gaussian elimination, then each f-evaluation requires $O(N^3)$ flops because the underlying LU factorization must be recomputed from scratch for each λ.

  If many f-evaluations are required, then a better approach is to rely on a similarity transformation such as the Schur or Hessenberg decomposition:

$$(1.3) \qquad\qquad\qquad Q^H A Q = T.$$

Here $Q \in \mathbb{C}^{N \times N}$ is unitary, $T \in \mathbb{C}^{N \times N}$ is upper triangular, quasi-triangular or Hessenberg, depending on whether A has complex eigenvalues and depending on whether the real or complex Schur (or Hessenberg) decomposition is used. Once this $O(N^3)$ "investment" is performed, then

$$f(\lambda) = \tilde{d}^T (T - \lambda I_N)^{-1} \tilde{b} \qquad\qquad \tilde{b} = Q^T b, \ \tilde{d} = Q^T d$$

can be evaluated in just $O(N^2)$ flops. In practice, one typically invokes the Hessenberg decomposition because it is cheaper, or the real Schur decomposition because it permits the handling of the complex eigenvalue case with real arithmetic.

  Applying these ideas to (1.1) we first compute the Schur decompositions

$$(1.4) \qquad\qquad Q^{(i)H} A^{(i)} Q^{(i)} \ = \ T^{(i)} \qquad i = 1{:}p,$$

a calculation that requires $O(n_1^3 + \cdots + n_p^3)$ flops. If

$$Q = Q^{(p)} \otimes \cdots \otimes Q^{(1)},$$

then Q is unitary and

$$Q^H \left(A^{(p)} \otimes \cdots \otimes A^{(1)} \right) Q \ = \ T^{(p)} \otimes \cdots \otimes T^{(1)}.$$

Thus, (1.1) transforms to

$$(1.5) \qquad\qquad \left(T^{(p)} \otimes \cdots \otimes T^{(1)} - \lambda I_N \right) y = c$$

where $y \in \mathbb{R}^N$ and $c \in \mathbb{R}^N$ are defined by

$$(1.6) \qquad\qquad x \ = \ \left(Q^{(p)} \otimes \cdots \otimes Q^{(1)} \right) y$$

and

$$(1.7) \qquad\qquad c \ = \ \left(Q^{(p)} \otimes \cdots \otimes Q^{(1)} \right)^H b.$$

If the Kronecker structure is exploited, then the computations for x and c require $O(N(n_1 + \cdots + n_p))$ flops. If the complex Schur decomposition is used, then the

resulting system (1.5) is triangular, and we show that it can also be solved in $O(N(n_1 + \cdots + n_p))$ flops. If the real Schur decomposition is used, then the Kronecker product in (1.5) has a complicated structure. In this case, we invoke the complex Schur decomposition to deal with the 2-by-2 bumps in each of the $T^{(i)}$. Regardless, the system (1.5) is an example where introducing complex arithmetic to solve a real problem is more advantageous. Our main contribution is to show that we can solve (1.5) "just as fast" where the $T^{(i)}$ are either upper triangular or upper quasi-triangular. In both cases, complex operations are used to solve the problem.

Thus, with our new method in place, the overall solution process (1.4)-(1.7) requires just $O(N(n_1 + \cdots + n_p))$ flops to execute. To put this in perspective, $O(N^2)$ flops are typically needed for N-by-N triangular system solving and $O(N^3)$ flops for the preliminary factorization. Note that we are assuming that the Schur decompositions in (1.4) are insignificant. Exceptions occur, for example, when $p = 2$ and $n_1 >> n_2$.

We stress that it is the presence of the shift λ in (1.5) that creates the problem. If $\lambda = 0$, then we have an easy factorization of the matrix of coefficients. Indeed, if the $T^{(i)}$ are upper triangular,

$$\left(T^{(p)} \otimes \cdots \otimes T^{(1)} \right) = \prod_{i=1}^{p} \left(I_{\rho_i} \otimes T^{(i)} \otimes I_{\mu_i} \right).$$

where $\rho_i = n_{i+1} \cdots n_p$ and $\mu_i = n_1 \cdots n_{i-1}$ for $i = 1{:}p$. A sequence of triangular system solves can then be used to obtain y:

(1.8)
$$
\begin{aligned}
&y \leftarrow c \\
&\text{for } i = 1{:}p \\
&\quad y \leftarrow \left(I_{\rho_i} \otimes T^{(i)} \otimes I_{\mu_i} \right)^{-1} y \\
&\text{end}
\end{aligned}
$$

This implementation of back-substitution requires $N(n_1 + \cdots + n_p)$ flops.

Unfortunately, if $\lambda \neq 0$ then we are stranded without a "Kronecker-friendly" factorization for $\left(T^{(p)} \otimes \cdots \otimes T^{(1)} - \lambda I_N \right)$. However, we can implement a recursive back-substitution procedure involving the Schur decomposition so that (1.5) can be solved as fast as (1.8).

Our presentation is structured as follows. In §2 we review relevant properties of the Kronecker product. To motivate our general procedure for both the triangular and quasi-triangular case, we consider the $p = 2$ case in §3. In section §4 we present the algorithm for general p using both the complex Schur decomposition and the real Schur decomposition. Numerical behavior and various performance and implementation issues are discussed at the end in §5. Finally, the error analysis is presented in the appendix.

2. Some Properties of the Kronecker Product. We review a few essential facts about the Kronecker product. Details and proofs can be found in [4].

Matrix computations that involve the Kronecker product require an understanding of the vec and reshape operators. If $Z \in \mathbb{R}^{m \times n}$, then the vec operator is defined

by

$$\text{vec}(Z) = \left[\begin{array}{c} Z(:,1) \\ \vdots \\ Z(:,n) \end{array} \right] \in \mathbb{R}^{mn}.$$

In other words, $\text{vec}(Z)$ is a vector obtained by stacking the columns of Z.

The $\texttt{reshape}$ operator is a more general way of rearranging the entries in a matrix. (It is also a built-in MATLAB function.) If $z \in \mathbb{R}^{mn}$ then $Z = \texttt{reshape}(z, m, n)$ is the m-by-n matrix defined by

$$Z(:,j) = z(1 + (j-1)m{:}jm) \qquad j = 1{:}n.$$

For example, if $m = 3$ and $n = 5$, then

$$\texttt{reshape}(z, 3, 5) = \left[\begin{array}{ccccc} z_1 & z_4 & z_7 & z_{10} & z_{13} \\ z_2 & z_5 & z_8 & z_{11} & z_{14} \\ z_3 & z_6 & z_9 & z_{12} & z_{15} \end{array} \right] = Z.$$

Thus, $\texttt{reshape}(z, m, n)$ makes a matrix out of z by using its components to "fill up" an m-by-n array in column-major order. We also use $\texttt{reshape}$ to build new matrices from the components of a given matrix. If $Z \in \mathbb{R}^{m_1 \times n_1}$ and $m_2 n_2 = m_1 n_1$, then $\texttt{reshape}(Z, m_2, n_2)$ is the m_2-by-n_2 matrix $\texttt{reshape}(\text{vec}(Z), m_2, n_2)$.

If F, G, H, and K are matrices and the multiplications FH and GK are defined, then $(F \otimes G)(H \otimes K) = FH \otimes GK$. Moreover, $(F \otimes G)^{-1} = (F^{-1} \otimes G^{-1})$ and $(F \otimes G)^T = F^T \otimes G^T$, assuming in the former case that F and G are nonsingular.

In general, if $F \in \mathbb{R}^{m \times m}$ and $G \in \mathbb{R}^{n \times n}$ then $F \otimes G \neq G \otimes F$. However, if we define the permutation matrix $\Pi_{n,nm} \in \mathbb{R}^{mn \times mn}$ by

$$\Pi_{n,nm}^T x = \left[\begin{array}{c} x(1{:}n{:}nm) \\ x(2{:}n{:}nm) \\ \vdots \\ x(n-1{:}n{:}nm) \end{array} \right],$$

then it can be shown that

$$\Pi_{n,nm}^T (F \otimes G) \Pi_{n,nm} = G \otimes F.$$

The matrix $\Pi_{n,nm}$ is called the *vec permutation matrix* and its action on a vector is very neatly described in terms of the $\texttt{reshape}$ operation:

(2.1) $\qquad y = \Pi_{n,mn}^T x \quad \Leftrightarrow \quad \texttt{reshape}(y, m, n) = \texttt{reshape}(x, n, m)^T$

(2.2) $\qquad y = \Pi_{n,mn} x \quad \Leftrightarrow \quad \texttt{reshape}(y, n, m) = \texttt{reshape}(x, m, n)^T$

Note that $y = \Pi_{2,52} x$ is the perfect shuffle of the "card deck" $x \in \mathbb{R}^{52}$. We mention that if x (and y) are complex, then (2.1) and (2.2) apply exactly as they are specified – the transpose is *not* replaced by a conjugate transpose.

The $\texttt{vec}$ operator enables us to identify certain matrix-vector products as matrix-matrix products. In particular, if $F \in \mathbb{R}^{m \times m}$, $G \in \mathbb{R}^{n \times n}$, and $X \in \mathbb{R}^{n \times m}$, then it can be shown that

(2.3) $\qquad Y = GXF^T \quad \Leftrightarrow \quad \texttt{vec}(Y) = (F \otimes G)\,\texttt{vec}(X).$

For matrix-vector products of the form

$$(2.4) \qquad y = (F_p \otimes \cdots \otimes F_1)\, x \qquad F_i \in \mathbb{R}^{n_i \times n_i}$$

it is convenient to make use of the factorization

$$(2.5) \qquad F_p \otimes \cdots \otimes F_1 \;=\; M_p \cdots M_1$$

where

$$(2.6) \qquad M_i = \Pi_{n_i, N}^T \left(I_{N/n_i} \otimes F_i \right)$$

and $N = n_1 \cdots n_p$. This result can be found in [4, p.153] where it is exploited in connection with high-dimensional FFTs. In practice, here is how one typically computes the vector y in (2.4):

$$(2.7) \qquad \begin{aligned} &Z \leftarrow x \\ &\text{for } i = 1{:}p \\ &\qquad Z \leftarrow (F_i \cdot \texttt{reshape}(Z, n_i, N/n_i))^T \\ &\text{end} \\ &y \leftarrow \texttt{reshape}(Z, N, 1) \end{aligned}$$

The ith pass through the loop requires $(2n_i^2)(N/n_i) = 2Nn_i$ flops so the overall computation involves $2N(n_1 + \cdots n_p)$ flops.

We mention that a similar process can be used to solve

$$(F_p \otimes \cdots \otimes F_1)\, x = d.$$

From (2.5) and (2.6) it follows that

$$(F_p \otimes \cdots \otimes F_1)^{-1} \;=\; F_p^{-1} \otimes \cdots \otimes F_1^{-1} \;=\; M_1^{-1} \cdots M_p^{-1}$$

where

$$M_i^{-1} = \left(I_{N/n_i} \otimes F_i^{-1} \right) \Pi_{n_i, N}$$

and so we obtain

$$(2.8) \qquad \begin{aligned} &B \leftarrow d \\ &\text{for } i = p{:} - 1{:}1 \\ &\qquad B \leftarrow F_i^{-1}\, \texttt{reshape}(B, N/n_i, n_i)^T \\ &\text{end} \\ &x \leftarrow \texttt{reshape}(B, N, 1) \end{aligned}$$

If the F_i are triangular, then the i-th pass through the loop requires $n_i^2(N/n_i) = Nn_i$ flops.

3. The $p = 2$ Case. To motivate the proposed new method for (1.5) for the triangular and quasi-triangular case, we first consider the special case when $p = 2$. That is, for $F \in \mathbb{R}^{m \times m}$ and $G \in \mathbb{R}^{n \times n}$,

(3.1) $$(F \otimes G - \lambda I)y = c.$$

Using (2.3), one can rewrite (3.1) as the real discrete-time Sylvester matrix equation

$$GY F^T - \lambda Y = C,$$

where $Y = \texttt{reshape}(y, n, m)$ and $C = \texttt{reshape}(c, n, m)$. As we mentioned earlier, a block procedure for solving the real discrete-time Sylvester matrix equation is described in [2]. In this section, we describe the details of solving (3.1) in a way that facilitates the presentation of the general, $p > 2$ algorithm. We begin with a small particular problem, $(F \otimes G - \lambda I_{3n}) y = c$, where

$$F = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ 0 & f_{22} & f_{23} \\ 0 & 0 & f_{33} \end{bmatrix}$$

and $G \in \mathbb{R}^{n \times n}$ is upper triangular. The shifted Kronecker system has the form

$$\begin{bmatrix} f_{11}G - \lambda I_n & f_{12}G & f_{13}G \\ 0 & f_{22}G - \lambda I_n & f_{23}G \\ 0 & 0 & f_{33}G - \lambda I_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

where $y_i \in \mathbb{R}^n$ and $c_i \in \mathbb{R}^n$ for $i = 1{:}3$. Assume that the system is nonsingular. The first step is to solve the n-by-n triangular system

$$(f_{33}G - \lambda I_n) y_3 = c_3$$

for y_3. By substituting y_3 into the first two equations we obtain

(3.2) $$\begin{bmatrix} f_{11}G - \lambda I_n & f_{12}G \\ 0 & f_{22}G - \lambda I_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \tilde{c}_1 \\ \tilde{c}_2 \end{bmatrix}$$

where $\tilde{c}_i = c_i - f_{i3}Gy_3$, $i = 1{:}2$. The vectors y_3 and Gy_3 require $O(n^2)$ flops. This process is then repeated to render y_2, and y_1 in turn.

Note from the example that if G is quasi-triangular (or even Hessenberg), then the systems involving the $(f_{ii}G - \lambda I_n)$ still just require $O(n^2)$ flops to solve. However, if F is upper quasi-triangular, then there is a more serious complication. To illustrate let us examine the system $(F \otimes G - \lambda I_{4n}) y = c$ where

$$F = \begin{bmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ 0 & f_{22} & f_{23} & f_{24} \\ 0 & f_{32} & f_{33} & f_{34} \\ 0 & 0 & 0 & f_{44} \end{bmatrix}$$

and $G \in \mathbb{R}^{n \times n}$ is upper quasi-triangular. In this case the shifted Kronecker system has the form

$$
\begin{bmatrix}
f_{11}G - \lambda I_n & f_{12}G & f_{13}G & f_{14}G \\
0 & f_{22}G - \lambda I_n & f_{23}G & f_{24}G \\
0 & f_{32}G & f_{33}G - \lambda I_n & f_{34}G \\
0 & 0 & 0 & f_{44}G - \lambda I_n
\end{bmatrix}
\begin{bmatrix}
y_1 \\ y_2 \\ y_3 \\ y_4
\end{bmatrix}
=
\begin{bmatrix}
c_1 \\ c_2 \\ c_3 \\ c_4
\end{bmatrix}
$$

where $y_i \in \mathbb{R}^n$ and $c_i \in \mathbb{R}^n$ for $i = 1{:}4$. Assume that the system is nonsingular. The first step is to solve the n-by-n quasi-triangular system

$$
(f_{44}G - \lambda I_n)\, y_4 = c_4
$$

for y_4. Substituting this into the above system reduces it to

$$
(3.3) \qquad
\begin{bmatrix}
f_{11}G - \lambda I_n & f_{12}G & f_{13}G \\
0 & f_{22}G - \lambda I_n & f_{23}G \\
0 & f_{32}G & f_{33}G - \lambda I_n
\end{bmatrix}
\begin{bmatrix}
y_1 \\ y_2 \\ y_3
\end{bmatrix}
=
\begin{bmatrix}
\tilde{c}_1 \\ \tilde{c}_2 \\ \tilde{c}_3
\end{bmatrix}
$$

where $\tilde{c}_i = c_i - f_{i4}Gy_4$. The vectors y_4 and Gy_4 require $O(n^2)$ flops. Next we solve the block 2-by-2 system

$$
(3.4) \qquad
\begin{bmatrix}
f_{22}G - \lambda I_n & f_{23}G \\
f_{32}G & f_{33}G - \lambda I_n
\end{bmatrix}
\begin{bmatrix}
y_2 \\ y_3
\end{bmatrix}
=
\begin{bmatrix}
\tilde{c}_2 \\ \tilde{c}_3
\end{bmatrix}
$$

for y_2 and y_3. We are then left with a single system for y_1:

$$
(f_{11}G - \lambda I_n)y_1 = \tilde{c}_1 - f_{12}Gy_2 - f_{13}Gy_3.
$$

From this example the general plan is clear. At each stage we solve either an n-by-n system for a single y_i or a $2n$-by-$2n$ block system for a pair of y_i's. The results are then substituted into the remaining equations.

Now let us consider how to solve a system of the form (3.4). For concreteness, suppose

$$
G =
\begin{bmatrix}
\times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times \\
0 & 0 & \times & \times & \times \\
0 & 0 & 0 & \times & \times \\
0 & 0 & 0 & \times & \times
\end{bmatrix}.
$$

It is easy to verify that the 2-by-2 block matrix of coefficients in (3.4) has the form

$$
(3.5) \qquad
\left[
\begin{array}{ccccc|ccccc}
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & 0 & \times & \times & \times & 0 & 0 & \times & \times & \times \\
0 & 0 & 0 & \times & \times & 0 & 0 & 0 & \times & \times \\
0 & 0 & 0 & \times & \times & 0 & 0 & 0 & \times & \times \\
\hline
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\
0 & 0 & \times & \times & \times & 0 & 0 & \times & \times & \times \\
0 & 0 & 0 & \times & \times & 0 & 0 & 0 & \times & \times \\
0 & 0 & 0 & \times & \times & 0 & 0 & 0 & \times & \times
\end{array}
\right] = S.
$$

Since $S = F(2:3, 2:3) \otimes G - \lambda I_{10}$, we can reverse the order of the Kronecker factors via a permutation as discussed in §2:

$$\Pi_{5,10}^T S \Pi_{5,10} = G \otimes F(2:3, 2:3) - \lambda I_{10} = \left[\begin{array}{cccc|cc|cccc} \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \hline 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times \end{array} \right].$$

Extrapolating from this example it is clear that a block system like (3.4) can be solved in $O(n^2)$ flops by permuting it into a block triangular system with diagonal blocks that are either 2-by-2 or 4-by-4.

    We are now in a position to formulate a complete algorithm for the problem $(F \otimes G - \lambda I_{mn})y = c$ when $F \in \mathbb{R}^{m \times m}$ and $G \in \mathbb{R}^{n \times n}$ are upper quasi-triangular. If

$$F = \left[\begin{array}{cccc} F_{11} & F_{12} & \cdots & F_{1r} \\ 0 & F_{22} & \cdots & F_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F_{rr} \end{array} \right]$$

with 1-by-1 and 2-by-2 diagonal blocks, then $(F \otimes G - \lambda I_{mn})y = c$ has the form

$$\left[\begin{array}{cccc} F_{11} \otimes G - \lambda I_\ell & F_{12} \otimes G & \cdots & F_{1r} \otimes G \\ 0 & F_{22} \otimes G - \lambda I_\ell & \cdots & F_{2r} \otimes G \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & F_{rr} \otimes G - \lambda I_\ell \end{array} \right] \left[\begin{array}{c} y_1 \\ y_2 \\ \vdots \\ y_r \end{array} \right] = \left[\begin{array}{c} c_1 \\ c_2 \\ \vdots \\ c_r \end{array} \right],$$

where $\ell = n$ if F_{ii} is 1-by-1 and $\ell = 2n$ if F_{ii} is 2-by-2 for $i = 1{:}r$. The overall back-substitution process then looks like this:

(3.6)

        for $k = r: -1:1$
            if F_{kk} is 1-by-1
                Solve $(F_{kk}G - \lambda I_n)y_k = c_k$ for $y_k \in \mathbb{R}^n$
                $z \leftarrow Gy_k$
                $c_i \leftarrow c_i - F_{ik}z, \;\; i = 1{:}k - 1$
            else
                Solve $(F_{kk} \otimes G - \lambda I_{2n})y_k = c_k$ for $y_k \in \mathbb{R}^{2n}$
                $z \leftarrow (I_2 \otimes G)y_k$
                $c_i \leftarrow c_i - (F_{ik} \otimes I_n)z, \; , \; i = 1{:}k - 1$
            end
        end

Thus, each pass through the loop we solve an n-by-n quasi-triangular system or a $2n$-by-$2n$ block triangular system obtained via permutation. The exact flop count depends upon the number of 2-by-2 blocks along the diagonals of F and G, i.e., the number of complex conjugate eigenvalue pairs that these matrices have. But regardless, the volume of computation is $O(mn(m+n))$.

4. The General Algorithm. Observe that Algorithm 3.6 could be a solution framework for the general $(T^{(p)} \otimes \cdots \otimes T^{(1)})y = c$ problem if we set

$$
\begin{aligned}
F &= T^{(p)} \\
G &= T^{(p-1)} \otimes \cdots \otimes T^{(1)} \\
m &= n_p \\
n &= n_1 \cdots n_{p-1}
\end{aligned}
$$

The "solve" steps in (3.6) become recursive calls. If the $T^{(i)}$ are all upper triangular, then F_{kk} is 1-by-1 and Algorithm 3.6 can be easily extended for general p. However, if the $T^{(i)}$ are quasi-triangular then F_{kk} is 2-by-2 and the system $(F_{kk} \otimes G - \lambda I_{2n})y_k = c_k$ has the form

$$
\left(F_{kk} \otimes T^{(p-1)} \otimes \cdots \otimes T^{(1)} - \lambda I_m \right) y_k = c_k.
$$

If we use the methods of the previous section, we can permute this system to obtain

$$
(4.1) \qquad \left(T^{(p-1)} \otimes \cdots \otimes T^{(1)} \otimes F_{kk} - \lambda I_m \right) \tilde{y}_k = \tilde{c}_k.
$$

However, the permute-to-block-triangular-form approach that we illustrated in §3 is much less appealing when we consider the general p case. If G is itself a Kronecker product, e.g., $T^{(p-1)} \otimes \cdots \otimes T^{(1)}$, then its structure is adversely scrambled when we permute S in (3.5).

For this reason, if we are confronted with a system of the form

$$
(4.2) \qquad\qquad (\alpha \otimes G - \lambda I)\, y = c
$$

where

$$
\alpha = \left[\begin{array}{cc} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{array} \right]
$$

has complex eigenvalues, then we compute the *complex* 2-by-2 Schur decomposition

$$
Q^H \alpha Q = \left[\begin{array}{cc} s_{11} & s_{12} \\ 0 & s_{22} \end{array} \right].
$$

Equation (4.2) transforms to

$$
\left(\left[\begin{array}{cc} s_{11} & s_{12} \\ 0 & s_{22} \end{array} \right] \otimes G - \lambda I \right) z = d
$$

where $z = (Q^H \otimes I)y$ and $d = (Q^H \otimes I)c$. This can be solved recursively when G is a Kronecker product. The (real) solution to the original system is then prescribed by $y = (Q \otimes I)z$.

Therefore, if the $T^{(i)}$ are upper quasi-triangular, extending Algorithm 3.6 for general p involves creating an input parameter, α, that can be either a 1-by-1 or 2-by-2 matrix. If α is 2-by-2 we compute its complex Schur decomposition and solve

$$\left(\left[\begin{array}{cc} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{array} \right] \otimes T^{(p-1)} \otimes \cdots \otimes T^{(1)} - \lambda I \right) y = c$$

recursively.

Of course, the hassle associated with the 2-by-2 bumps can be avoided altogether if the complex Schur decompositions $Q^{(i)H} A Q^{(i)} = T^{(i)}$ are computed right at the start. However, the proposed strategy is preferred because it restricts complex arithmetic to diagonal block subproblems. In pseudo-MATLAB our algorithm, `KPShiftSolve`, is given in Figure 4.1.

To assess the volume of the computation, let ν_p be the number of flops required by a call to `KPShiftSolve` when the matrix of coefficients involves a p-fold Kronecker product. Ignoring low-order terms,

$$(4.3) \qquad \nu_p = \begin{cases} 1.5n_1^2 & \text{if } p = 1 \\ \\ n_p \nu_{p-1} + n_1 \cdots n_p (n_1 + \cdots + n_p) & \text{if } p > 1 \end{cases}$$

Of course, the exact flop count depends on the number of complex eigenvalues of the $T^{(i)}$. We mentioned that an alternative, but more costly, algorithm involves computing the complex Schur decompositions of the $T^{(i)}$ from the start. If $p = 1$, then solving $(T_1 - \lambda I)y = c$ in complex arithmetic requires $6.5n_1^2$ as compared with (4.3). When $p > 1$, the cost of the update (2.7) also increases by a factor of 6 using complex arithmetic. Hence, the exact flop count of `KPShiftSolve` lies between what is specified in (4.3) and the bounds given above when complex arithmetic is used for the entire process.

If $n_1 = \cdots = n_p \equiv n$ in (4.3), then it can be shown that

$$\nu_p = \frac{1 + p + p^2}{2} n^{p+1} \;=\; \frac{1 + p + p^2}{2} N n$$

Things are even more complicated if the n_i vary. For example, if the $T^{(i)}$ are triangular and real, then having $n_1 \leq \cdots \leq n_p$ can is more advantageous than having $n_1 \geq \cdots \geq n_p$ to reduce the number of recursive calls. For quasi-triangular $T^{(i)}$, the flop count, vectorization properties, and recursion overheads depend upon the size ordering and the number of 2-by-2 bumps in each $T^{(i)}$.

5. Implementation Issues and Performance. `KPShiftSolve` can be made more efficient in two ways. First, the $T^{(i)}$ should be sorted so that $T^{(i+1)}$ has fewer 2-by-2 bumps than $T^{(i)}$, $i = 1{:}p - 1$ so there are fewer recursive calls. This can be accomplished via the perfect shuffle explained in §2 and reduces the overall flop count. Second, instead of computing the real Schur form of $T^{(1)}$, we need only compute the cheaper Hessenberg decomposition. To appreciate why this is sufficient consider the $p = 2$ example at the start of §3. If G is upper Hessenberg then the linear systems with coefficient matrices $f_{ii}G - \lambda I$ that arise during the back-substitution process (3.2) are also upper Hessenberg. Hessenberg systems can be solved just as fast as quasi-triangular systems [1, p.155].

MATLAB codes for the algorithms discussed in this paper are available at
$$\texttt{http://www.cam.cornell.edu/}{\sim}\texttt{carlam/}.$$

function $y = $ KPShiftSolve$(T, n, c, \lambda, \alpha)$

% T is a length p cell array and $n = (n_1, \ldots, n_p)$. Assume that the
% i-th element of T is the upper quasi-triangular matrix T_i.
% Set $N = n_1 \cdots n_p$. α is a scalar or a 2-by-2 matrix. If λ is a real scalar
% and $c \in \mathbb{R}^N$, then $y \in \mathbb{R}^N$ solves $(\alpha \otimes T_p \otimes \cdots \otimes T_1 - \lambda I)y = c$ assuming that
% the system is nonsingular. $\alpha = 1$ is the default value when not specified.

$p = $ length(n); $N = $ prod(n);

if $\alpha \in \mathbb{R}$

    $T_p = \alpha T_p$

    if $p == 1$

        Solve $(T_1 - \lambda I_{n_1})y = c$ for y.

    else

        $y = $ zeros$(N, 1)$; $m_p = N/n_p$; $i = n_p$;

        while $(i \geq 1)$

            if $i > 1$ & $T_p(i, i - 1) \neq 0$ (T_p has a 2-by-2 bump)

                $idx = 1 + (i - 2)m_p : im_p$

                $y(idx) = $ KPShiftSolve$(T, n(1{:}p - 1), c(idx), \lambda, T_p(i - 1{:}i, i - 1{:}i))$

                $z_1 = (T_{p-1} \otimes \cdots \otimes T_1) \cdot y(idx(1){:}(i - 1)m_p)$ (Invoke (2.7))

                $z_2 = (T_{p-1} \otimes \cdots \otimes T_1) \cdot y(1 + (i - 1)m_p{:}idx(end))$ (Invoke (2.7))

                for $j = 1{:}i - 2$

                    $jdx = 1 + (j - 1)m_p : jm_p$

                    $c(jdx) = c(jdx) - T_p(j, i - 1)z_1 - T_p(j, i)z_2$

                end

                $i = i - 2$

            else   (T_p does not have a 2-by-2 bump)

                $idx = 1 + (i - 1)m_p : im_p$

                $y(idx) = $ KPShiftSolve$(T, n(1{:}p - 1), c(idx), \lambda, T_p(i, i))$

                $z = (T_{p-1} \otimes \cdots \otimes T_1)\, y(idx)$ (Invoke (2.7))

                for $j = 1{:}i - 1$

                    $jdx = 1 + (j - 1)m_p : jm_p$

                    $c(jdx) = c(jdx) - T_p(j, i)z$

                end

                $i = i - 1$

            end

        end

    end

else $(\alpha \in \mathbb{R}^{2 \times 2})$

    Compute Q unitary, S upper triangular so that $Q^H \alpha Q = S$

    $d = (Q^H \otimes I)c$; $T_{p+1} = S$; $n_{p+1} = 2$;

    $z = $ KPShiftSolve$(T, n, d, \lambda, 1)$

    $y = (Q \otimes I)z$; $y = $ real(y);

end

FIG. 4.1. *Pseudo-*MATLAB *code for* KPShiftSolve

The reader interested in details, especially as they concern the recursion, should study the codes directly.

With respect to the roundoff properties of the algorithm, the computed solution $\hat{x}$ can be shown to solve

$$(5.1) \qquad \left(A^{(p)} \otimes \cdots \otimes A^{(1)} - \lambda I_N + E \right) \hat{x} = b$$

where for any p-norm

$$(5.2) \quad \| E \| \approx \mathbf{u} \left(\| A^{(p)} \| \cdots \| A^{(1)} \| + |\lambda| \right) \;=\; \mathbf{u} \left(\| A^{(p)} \otimes \cdots \otimes A^{(1)} \| + |\lambda| \right)$$

and $\mathbf{u}$ is the unit roundoff. See Appendix 1 for details.

We make a three comments related to (5.2). First, the explicit formation of the coefficient matrix involves rounding errors of the same magnitude as $\| E \|$. Second, as with any shifted, nonsymmetric linear system, there is not much we can say about the forward stability in $\hat{x}$ because the connection between the condition and the shift parameter is nontrivial. Finally, if $\kappa_p(\cdot)$ denotes the p-norm condition, then $\kappa_p(A^{(p)} \otimes \cdots \otimes A^{(1)}) = \kappa_p(A^{(p)}) \cdots \kappa_p(A^{(1)})$. Thus, modest ill-conditioning among the $A^{(i)}$ compounds to severe ill-conditioning in the Kronecker product.

6. Conclusion. We have presented an algorithm that solves the shifted system

$$\left(A^{(p)} \otimes \cdots \otimes A^{(1)} - \lambda I_N \right) x = b$$

where $A^{(i)} \in \mathbb{R}^{n_i \times n_i}$ for $i = 1{:}p$ and $N = n_1 \cdots n_p$. Our algorithm involves taking the real Schur decompositions to convert this system to a quasi-triangular system and uses a recursive block back-substitution procedure (`KPShiftSolve`). When a 2-by-2 bump is encountered in the leading coefficient matrix, the complex Schur decomposition is computed of the 2-by-2 matrix. This is faster than computing the complex Schur decompositions from the start. The error associated with our algorithm is no worse than the method of actually forming the Kronecker Product and using standard back-substitution.

Appendix A. Error Analysis. In this appendix, we establish (5.1) and (5.2). Recall the first step in our algorithm is to compute the (real) Schur decompositions of each $A^{(i)}$. Because of the results in [3], it suffices to show that if $\hat{y}$ is produced by `KPShiftSolve` then

$$(A.1) \qquad (T^{(p)} \otimes \ldots \otimes T^{(1)} - \lambda I + \Delta T)\hat{y} = c$$

$$(A.2) \qquad \| \Delta T \| \leq \delta_T (||T^{(p)}|| \ldots ||T^{(1)}|| + |\lambda|)$$

where δ_T is a modest multiple of the unit roundoff $\mathbf{u}$ and $T^{(i)}$ is either triangular or quasi-triangular for $i = 1{:}p$. To establish these results we first say something about the case $p = 2$. As in [3], we adopt the convention that all the δ's below are $O(\mathbf{u})$ in magnitude. In addition, the floating point result of a matrix calculation is indicated by $\mathrm{fl}(\cdot)$ and we use "hat" notation to represent computed quantities.

LEMMA A.1. *Let $F \in \mathbb{R}^{m \times m}$ be upper triangular, $G \in \mathbb{R}^{n \times n}$ be quasi-upper triangular, $c \in \mathbb{R}^{mn}$, and $\lambda \in \mathbb{R}$. If $\hat{y}$ is obtained by using `KPShiftSolve` to solve $(F \otimes G - \lambda I)y = c$, then there exists $E \in \mathbb{R}^{mn \times mn}$ such that*

$$(A.3) \qquad (F \otimes G - \lambda I + E)\hat{y} = c,$$

$$(A.4) \qquad \| E \| \leq \delta_E (\| F \| \| G \| + |\lambda|).$$

Proof. The proof is by induction on the dimension of F. If $m = 1$ then we solve $(f_{11}G - \lambda I_n)y = c$. This involves first forming $M = f_{11}G - \lambda I_n$ and then solving $My = c$ using back-substitution. Accounting for the rounding error associated with forming M, there exists H_1 such that

(A.5) $\hat{M} = \mathrm{fl}(M) = f_{11}G - \lambda I + H_1$ $\| H_1 \| \le \delta_1(|f_{11}| \cdot \| G \| + |\lambda|)$.

Next, the computed solution to the triangular system satisfies

(A.6) $$(\hat{M} + H_2)\hat{y} = c$$

(A.7) $$\| H_2 \| \le \delta_2 \| \hat{M} \| \le \delta_3(|f_{11}| \cdot \| G \| + |\lambda|)$$

where $\delta_3 = \delta_1 + \delta_2$ (see [1, p.89]). Combining (A.5)-(A.7) and setting $E = H_1 + H_2$ completes the proof when $m = 1$.

Now suppose (A.3), (A.4) hold for all $k < m$. Partition the system as

$$\begin{bmatrix} f_{11}G - \lambda I & F_{12} \otimes G \\ 0 & F_{22} \otimes G - \lambda I \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

where $F_{12} = F(1, 2{:}m)$, $F_{22} = F(2{:}m, 2{:}m)$ and y_1, y_2, c_1, c_2 are appropriate blockings of y and c, respectively. By induction, $\hat{y}_2$ solves

$$(F_{22} \otimes G - \lambda I + E_{22})\hat{y}_2 = c_2$$

with

(A.8) $$\| E_{22} \| \le \delta_4(\| F_{22} \| \| G \| + |\lambda|).$$

The next step is to solve for y_1 with

$$(f_{11}G - \lambda I)y_1 = \mathrm{fl}(c_1 - (F_{12} \otimes G)\hat{y}_2).$$

The computations associated with the update $d_1 = c_1 - (F_{12} \otimes G)\hat{y}_2$ satisfy

$$\hat{d}_1 = \mathrm{fl}(d_1) = c_1 - (F_{12} \otimes G + E_{12})\hat{y}_2 + \Delta c,$$

where

(A.9) $$\| E_{12} \| \le \delta_5 \| F_{12} \| \| G \|$$

(A.10) $$\| \Delta c \| \le \delta_6(\| c_1 \| + \| F_{12} \| \| G \| \| \hat{y}_2 \|).$$

Finally, we form $M = f_{11}G - \lambda I$ and solve $My_1 = d_1$ using back-substitution. Forming M gives

(A.11) $$\hat{M} = \mathrm{fl}(M) = f_{11}G - \lambda I + \Delta_1$$

(A.12) $$\| \Delta_1 \| \le \delta_7(|f_{11}| \cdot \| G \| + |\lambda|).$$

Then the computed solution to the triangular system $\hat{M}y_1 = \hat{d}_1$ satisfies

$$(\hat{M} + \Delta_2)\hat{y}_1 = \hat{d}_1$$

with

(A.13) $$\| \Delta_2 \| \le \delta_8 \| \hat{M} \| \le \delta_9(|f_{11} \cdot \| G \| + |\lambda|).$$

Let $E_{11} = \Delta_1 + \Delta_2$ and set

$$E = \left[\begin{array}{cc} E_{11} & E_{12} \\ 0 & E_{22} \end{array} \right].$$

The proof follows from (A.8), (A.9), (A.12), (A.13), and by setting $\delta_E = \delta_4 + \delta_5 + \delta_7 + \delta_9$.
□

Next, we address the error associated with computations when a 2-by-2 bump is encountered. For simplicity, we show this for $p = 2$.

LEMMA A.2. *Let α be a scalar or a 2×2 matrix with complex eigenvalues, and let T be an upper quasi-triangular $n \times n$ matrix. Let $m = n \dim(\alpha)$, $c \in \mathbb{R}^m$, and $\lambda \in \mathbb{R}$. If $\hat{y}$ is obtained from using KPShiftSolve to solve $(\alpha \otimes T - \lambda I)y = c$, then there exists $E \in \mathbb{R}^{m \times m}$ such that*

(A.14) $$(\alpha \otimes T - \lambda I_m + E)\hat{y} = c,$$

(A.15) $$\| E \| \le \delta_E(\| \alpha \| \| T \| + |\lambda|).$$

Proof. If α is 1×1, then the proof is completed by using Lemma A.1. If α is 2×2 then KPShiftSolve computes the complex Schur decomposition of α and then solves a block upper triangular system. Specifically, KPShiftSolve performs the following four steps:

1. Compute the complex Schur decomposition $Q^H \alpha Q = S$.
2. Form $d = (Q^H \otimes I)c$.
3. Solve the system $(S \otimes T - \lambda I)z = d$ for z using KPShiftSolve.
4. Set $y = (Q \otimes I)z$.

These steps are analogous to solving a linear system using the complex Schur decomposition. The Kronecker product structure does not affect the result and can be shown using the methods found in [3] and Lemma A.1.
□

Now that we have dealt with the error associated with solving the system when a 2-by-2 bump is encountered, we are ready to establish the error results for a Kronecker product of quasi-triangular matrices. In the next lemma, we present the results for $p = 2$.

LEMMA A.3. *Let F and G be a quasi-triangular matrices of size $m \times m$ and $n \times n$, respectively. Let $\lambda \in \mathbb{R}$ and $b \in \mathbb{R}^{mn}$. If KPShiftSolve is used to solve $(F \otimes G - \lambda I_{mn})y = c$ then there exists $E \in \mathbb{R}^{mn \times mn}$ such that the computed solution $\hat{y}$ solves*

(A.16) $$(F \otimes G - \lambda I + E)\hat{y} = c,$$

(A.17) $$\| E \| \le \delta_E(\| F \| \| G \| + |\lambda|).$$

Proof. The proof is similar to the proof of Lemma A.1 and is completed by induction on the dimension of F. Lemma A.2 is used when a 2-by-2 bump is encountered in F. ◻

We are now ready to establish the final result for general p. The following lemma establishes (A.1) and (A.2).

LEMMA A.4. *Let $T^{(1)}, \ldots, T^{(p)}$ be upper quasi-triangular $n_i \times n_i$ matrices, $i = 1, \ldots, p$. Let $\lambda \in \mathbb{R}$ and $b \in \mathbb{R}^{n_1 \cdots n_p}$. If KPShiftSolve is used to solve $(T^{(p)} \otimes \cdots \otimes T^{(1)} - \lambda I)y = c$, then there exists $\Delta T \in \mathbb{R}^{N \times N}$ where $N = n_1 \cdots n_p$ such that the computed solution $\hat{y}$ solves*

$$(T^{(p)} \otimes \cdots \otimes T^{(1)} - \lambda I + \Delta T)\hat{y} = c,$$

$$\| \Delta T \| \leq \delta_T(||T^{(p)}|| \cdots ||T^{(1)}|| + |\lambda|).$$

Proof. We prove this by induction on p. Lemma A.3 proves the base case when $p = 2$. Now assume Lemma A.4 holds for $p - 1$. To show this is true for p, we use induction on $\dim(A^{(p)})$. The proof follows by following the methods similar to the proof of Lemma A.1 when $p = 2$. The proof now easily follows from Lemma A.3 by setting $G = T^{(p-1)} \otimes \cdots \otimes T^{(1)}$. ◻

REFERENCES

[1] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Third Edition, Johns Hopkins University Press, Baltimore, MD, 1996.

[2] I. Jonsson and B. Kågström. Recursive blocked algorithm for solving triangular systems. II. Two-sided and generalized Sylvester and Lyapunov matrix equations. *ACM Trans. Math. Software*, 28 (4): 416-435, 2002.

[3] Carla D. Moravitz Martin and Charles F. Van Loan. *Solving Real Linear Systems with the Complex Schur Decomposition*, To appear: SIAM J. Matrix Anal. Appl. (2006).

[4] Charles F. Van Loan. *Computational Frameworks for the Fast Fourier Transform*, SIAM Publications, Philadelphia, 1992.