Algorithm 747: A Fortran Subroutine to Solve the Eigenvalue Assignment Problem for Multiinput Systems Using State Feedback

GEORGE MIMINIS and HELMUT ROTH
Memorial University of Newfoundland

The implementation of an algorithm for the computation of a state feedback for multiinput linear systems, resulting in a closed-loop matrix with a specified self-conjugate set of eigenvalues, is presented. The computation uses only real arithmetic, assigning complex conjugate eigenvalues in one double step. The implementation uses level-1 **BLAS** routines where possible. A brief description of the algorithm is also given.

Categories and Subject Descriptors: F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computations on matrices*; G.1.0 [**Numerical Analysis**]: General—*numerical algorithms*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*eigenvalues*; J.2 [**Computer Applications**]: Physical Sciences and Engineering—*aerospace*; *engineering*; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*economics*

General Terms: Algorithms

Additional Key Words and Phrases: Deflation, double steps, eigenvalue assignment, numerical efficiency, pole assignment

1. INTRODUCTION

The eigenvalue assignment problem arises in the design of stable linear systems. Such systems may be described by either the difference equation

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

or the differential equation

$$\dot{x}(t) = Ax(t) + Bu(t),$$

This work was supported by NSERC grant OGP0944 and by an NSERC undergraduate research award.
Authors' address: Department of Computer Science, Memorial University of Newfoundland, St. John's, NF, A1C 5S7, Canada; email: {miminis, helmut}@cs.mun.ca.

depending on whether they are observed in discrete or continuous time, respectively. In both cases, the $n \times n$ matrix A and the $n \times m$ matrix B are time invariant and describe the characteristics of the system. The state of the system is represented either at a discrete point (x_k) or at some particular instant in time ($x(t)$), and u represents the input to the system with the intent to control its state. The properties that are discussed in this introduction apply to both discrete- and continuous-time systems.

An important concept in the study of linear systems is *controllability*. A controllable system may be intuitively defined as a system that can be brought from an initial state to any desired state in a finite sequence of steps (or in finite time) using specified inputs (e.g., see Luenberger [1979, p. 276]). Another important concept in linear dynamic system theory is *stability*, defined intuitively next. Stability may be defined with respect to special vectors called *equilibrium points* (better known to mathematicians as *fixed*, *critical*, or *limit points*), which possess the property that, once the state of the system becomes equal to such a vector, it remains equal to it for all future time (e.g., see Luenberger [1979, p. 150]), given that the input applied to the system is constant. Suppose now that the state vector rests at some equilibrium point and that it is moved slightly away from that point. If the state vector tends to return to the equilibrium point, or at least if it does not keep moving further away from it, the system is stable; otherwise, the system is unstable (e.g., see Luenberger [1979, p. 154]). It may be shown that the stability of a linear system (discrete or continuous) depends on the eigenvalues of the matrix A (e.g., see Luenberger [1979, p. 157]). One way of stabilizing the system is by creating a matrix with a particular set of eigenvalues. This can be accomplished by computing an input using the current state of the system (*state feedback*). In discrete-time systems, for example, the input could be computed as

$$u_k = -Fx_k \tag{2}$$

with F being some $m \times n$ matrix. When combined with (2), Eq. (1) becomes

$$x_{k+1} = (A - BF)x_k. \tag{3}$$

To ensure the stability of the discrete-time system (3), the eigenvalues of the matrix $A - BF$ must be less than one in magnitude, whereas in continuous-time systems, the eigenvalues of $A - BF$ must have negative real parts (e.g., see Luenberger [1979, pp. 154–158]). For A, B real, the problem becomes one of choosing real F such that $A - BF$ has a specified self-conjugate set of eigenvalues. This is known as the *Eigenvalue* (or *Pole*) *Assignment Problem* (*EAP*), since by choosing F we are *assigning* the eigenvalues we need for the matrix $A - BF$.

The EAP can be shown to have a solution if and only if the system in (1) is controllable [Wonham 1967]. There is a good number of papers on the subject.

Some of the numerically oriented algorithms may be found in Kautsky et al. [1985], Miminis and Paige [1988], Patel and Misra [1984], Varga [1981], and Petkov et al. [1986]. A numerical comparison of these algorithms as well as other algorithms on the subject may be found in Miminis and Paige [1988], where there is also a rigorous rounding-error analysis, which establishes the numerical stability of the algorithm presented in the same paper. Also in Miminis and Paige [1988] it is stated that there is good possibility for the algorithm in Petkov et al. [1986] to be numerically stable. We believe the efficiency of the algorithm in Miminis and Paige [1988] and that of Petkov et al. [1986] is due to the fact that they are both QR-like algorithms (QR is an algorithm for the solution of the eigenproblem of a full matrix, e.g., see Golub and Van Loan [1989]). Here we implement an algorithm that substantially improves that of Miminis and Paige [1988]; it is presented in detail in Miminis and Paige [1994]. The algorithm in Miminis and Paige [1994], unlike that in Miminis and Paige [1988], assigns two eigenvalues at a time instead of just one. In this way, complex eigenvalues may be assigned as complex conjugate pairs, and thus, we completely avoid complex arithmetic. As a result, the algorithm presented in Miminis and Paige [1994] is more efficient than that in Miminis and Paige [1988] in terms of both time and space. It is also more realistic since it computes a real F, whereas that in Miminis and Paige [1988] produces, in general, a complex F.

In Section 2 we give a brief description of the algorithm and discuss some results of our experimenting with it. In Section 3 we give the computational cost of the algorithm, and in Section 4 we discuss some details of the implementation and its use.

We adopt the following notation: uppercase Roman letters are used to denote matrices; lowercase Roman letters denote column vectors and indices, while Greek characters denote scalars. The transpose of a matrix or vector is denoted by a superscript letter T (e.g., A^T or b^T). The notation $k = i : r : j$ means that k takes all of the values starting from i and j with step r; if $r = 1$, the above may be written as $i : j$ for simplicity. Given a matrix A, the MATLAB-like notation $A(r_1 : r_2, c_1 : c_2)$ will denote a submatrix of A that consists of rows r_1 through r_2 and columns c_1 through c_2. If, for example, all rows of A are included, the above submatrix may be written as $A(:, c_1 : c_2)$. For a product of matrices AB, the notation $AB(r_1 : r_2, c_1 : c_2)$, for example, will denote the corresponding submatrix of AB. The notation

$$R_2^T A R_1 = \begin{pmatrix} \times & \times \\ \underset{1}{\otimes} & \times \end{pmatrix} \begin{matrix} \\ 2 \end{matrix}$$

will denote that transformation R_1 (labeled 1) eliminates the $(2, 1)$-element of A into the $(2, 2)$-element. Then transformation R_2^T (labeled 2) simply combines the two rows of AR_1. Finally, O will denote a zero matrix or the big-O notation; e_i will denote the ith column of the identity matrix of size implied by the context, and $\mathbb{R}$ will denote the set of real numbers.

2. THE ALGORITHM

2.1 Description of the Algorithm

In this section we provide a brief description of the algorithm presented in Miminis and Paige [1994]. The algorithm is implemented in double and single precision by the **DMEVAS** and **SMEVAS** subroutines, respectively, which are described in Section 4. In the following we refer to both subroutines as **MEVAS**. Given a system (B, A), the algorithm in Miminis and Paige [1994] initially computes orthogonal matrices $T \in \mathbb{R}^{n \times n}$ and $U \in \mathbb{R}^{n \times m}$, such that (B, A) is transformed into a staircase form (e.g., see Miminis and Paige [1982], Paige [1981], and Van Dooren [1981]), as follows:

$$(B_1, A_1) \equiv T^T(B, A)\left(\begin{array}{c|c} U & \\ \hline & T \end{array}\right)$$

$$= \left(\begin{array}{c|ccccc} B_{11} & A_{11} & A_{12} & \cdots & A_{1,k-1} & A_{1k} \\ & A_{21} & A_{22} & \cdots & A_{2,k-1} & A_{2k} \\ & & A_{32} & \cdots & A_{3,k-1} & A_{3k} \\ & & & \ddots & \vdots & \vdots \\ & & & & A_{k,k-1} & A_{kk} \end{array}\right) \tag{4}$$

with $A_{ij} \in \mathbb{R}^{n_i \times n_j}$, where $n_1 = rank(B_{11})$ and $n_i = rank(A_{i,i-1})$, for $i = 2:(k-1)$ and $n_i \geq n_{i+1}$ for $i = 1:(k-1)$. The matrices B_{11} and $A_{i,i-1}$, $i = 2:(k-1)$, are of the form (O, R) with R nonsingular and upper triangular, with the possible exception of $A_{k,k-1}$, which is zero if the system is uncontrollable (see Paige [1981]). If the system is controllable, k is the *controllability index* of the system, and $n_k = rank(A_{k,k-1})$. Along with the **MEVAS** subroutines, we also provide two subroutines, **DSTAIR** and **SSTAIR**, that implement the staircase algorithm in double and single precision, respectively. In view of the fact that descriptions of the staircase algorithm may be found elsewhere, we give none here. Nevertheless, in Section 4 we do provide necessary instructions for the use of **STAIR**.

In the following we briefly describe how to compute $F_1 = U^T F T$ so that $A_1 - B_1 F_1 \equiv T^T(A - BUU^T F)T$ has specified eigenvalues when (B, A) is controllable. For simplicity we assume that $m = n_1 \equiv rank(B_{11})$; the case $m > n_1$ can easily be handled by setting the first $m - n_1$ rows of F_1 to zero (see Miminis and Paige [1994]). If, in (4), $n_1 - n_2 > 1$, then $s = 2\lfloor (n_1 - n_2)/2 \rfloor$ eigenvalues can be assigned "immediately." "Immediate" assignments, unlike regular assignments (which will be described in the sequel), do not need to transform the given problem in order to take effect. To see this, consider the following example, where $n = 5$, $m = 3$, $k = 3$, $n_1 = 3$, $n_2 = n_3 = 1$, with $\times$

representing a known element, and ★ representing an element to be determined:

$$A_1 - B_1F_1 = \left(\begin{array}{cc|ccc} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{array}\right) - \left(\begin{array}{ccc} \times & \times & \times \\ & \times & \times \\ & & \times \\ \hline & O & \end{array}\right)\left(\begin{array}{cc|ccc} \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star \end{array}\right) \tag{5}$$

$$= \left(\begin{array}{cc|ccc} \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star \\ \hline \star & \star & \star & \star & \star \\ & & \times & \times & \times \\ & & & \times & \times \end{array}\right).$$

Since B_{11} is upper triangular and nonsingular, we may easily solve the system of equations from (5),

$$\left(\begin{array}{cc} \times & \times \\ \times & \times \\ \times & \times \end{array}\right) - \left(\begin{array}{ccc} \times & \times & \times \\ & \times & \times \\ & & \times \end{array}\right)\left(\begin{array}{cc} \star & \star \\ \star & \star \\ \star & \star \end{array}\right) = \left(\begin{array}{cc} \zeta_1 & \eta \\ -\eta & \zeta_2 \\ 0 & 0 \end{array}\right),$$

with respect to the first two columns of F_1, where $\zeta_1 + \iota\eta$ and $\zeta_2 - \iota\eta$ are two specified eigenvalues, with $\iota^2 = -1$ and $\eta \neq 0 \Rightarrow \zeta_1 = \zeta_2$. Or, in general, we may solve

$$A_1(1:n_1, 1:s) - B_{11}F_1(:, 1:s) = \left(\begin{array}{ccc} Z_1 & & \\ & \ddots & \\ & & Z_{s/2} \\ \hline & O & \end{array}\right), \tag{6}$$

with respect to $F_1(:, 1:s)$, where

$$Z_\iota = \left(\begin{array}{cc} \zeta_{J1} & \eta_J \\ -\eta_J & \zeta_{J2} \end{array}\right)$$

corresponds to the pair of specified eigenvalues $\zeta_{J1} + \iota\eta_J$ and $\zeta_{J2} - \iota\eta_J$ with

$\eta_j \neq 0 \Rightarrow \zeta_{j1} = \zeta_{j2}$, for $j = 1:s/2$. After the solution of (6), $A_1 - B_1F_1$ in (5) has the following form:

$$A_1 - B_1F_1 = \left(\begin{array}{cc|ccc} \zeta_1 & \eta & \star & \star & \star \\ -\eta & \zeta_2 & \star & \star & \star \\ \hline & & \star & \star & \star \\ & & \times & \times & \times \\ & & & \times & \times \end{array}\right) \equiv \left(\begin{array}{cc|ccc} \zeta_1 & \eta & \star & \star & \star \\ -\eta & \zeta_2 & \star & \star & \star \\ \hline & & \multicolumn{3}{c}{A_2 - B_2F_2} \end{array}\right),$$

where $A_2 = A_1(s+1:n, s+1:n)$, $B_2 = B_1(s+1:n, s+1:m)$, and $F_2 = F_1(s+1:m, s+1:n)$. The algorithm now continues with the assignment of the remaining eigenvalues, to the system (B_2, A_2), by computing F_2. Note that $F_1(1:s, s+1:n)$ does not affect the eigenvalues; thus, it can be chosen to solve some other problem. For example, it is shown in Miminis and Paige [1994] that, if it is chosen so that

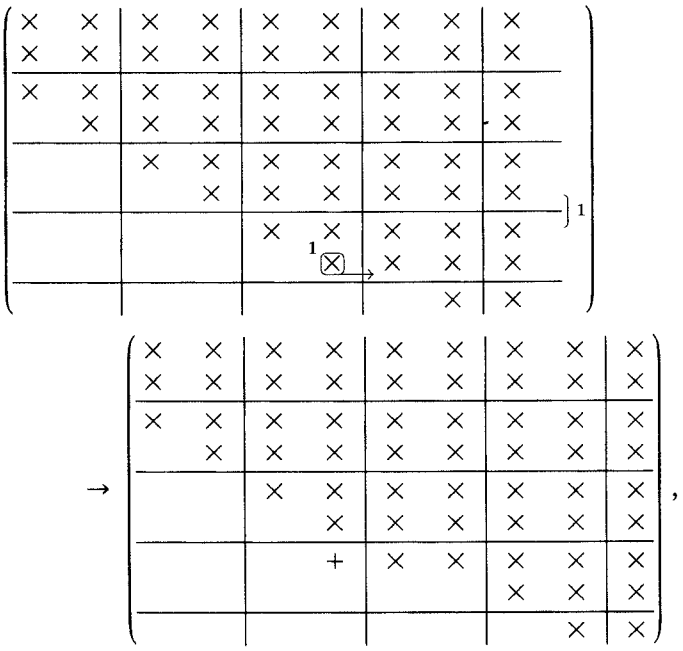
$$A_1 - B_1F_1 \equiv \left(\begin{array}{ccc|c} Z_1 & & & \\ & \ddots & & O \\ & & Z_{s/2} & \\ \hline \multicolumn{3}{c|}{O} & A_2 - B_2F_2 \end{array}\right),$$

the condition number of the eigenproblem of $A - BF$ is improved. In the **MEVAS** subroutines, however, it is simply set to zero, which decreases the magnitude of F. For more details about the description of the general case of the above procedure, see Miminis and Paige [1994]. "Immediate" assignments may also occur later in the algorithm; if that happens they will be handled in a similar way.

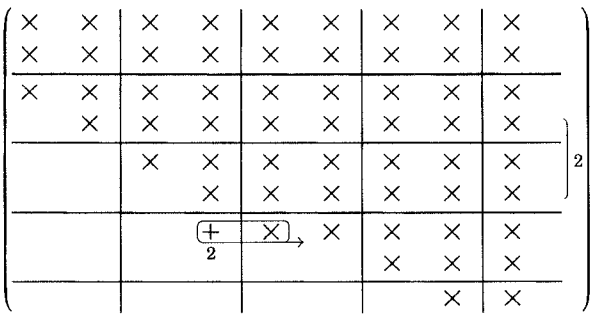
After the "immediate" assignment of the first s eigenvalues, we continue with a system (B_2, A_2) of the form (4), where the new n_1, n_2 satisfy either $n_1 = n_2$ or $n_1 = n_2 + 1$. In the case of $n_1 = n_2 + 1$, if a real eigenvalue exists, it may be assigned in an immediate step as above and then proceed with a system where the new n_1, n_2 satisfy $n_1 = n_2$. If there are no more real eigenvalues to be assigned, then the case $n_1 = n_2 + 1$ will be handled in the same way as the case $n_1 = n_2$, which is described next.

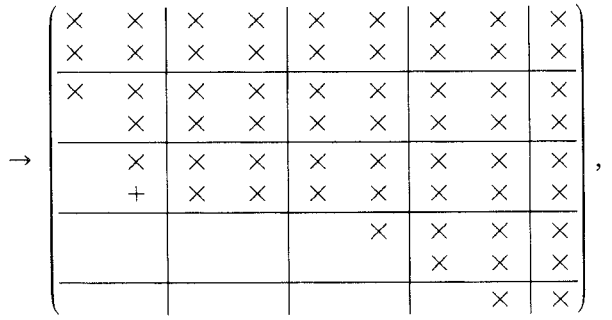
Let r be such that $n_2 = \cdots = n_r > n_{r+1}$, that is, the subdiagonal blocks $A_{21}, \ldots, A_{r,r-1}$ of A_1 in (4) are nonsingular upper triangular matrices, whereas $A_{r+1,r}$ is of the form (O, R), and set $q \equiv n_r$. In the following we consider two cases, $r > 2$ and $r = 2$, which will be discussed separately since they are treated differently. First, we consider the case $r > 2$, where the algorithm computes orthogonal matrix Q and performs the similarity transformation $Q^T(A_1 - B_1F_1)Q$. Details for the computation of Q along with the necessary theoretical justifications may be found in Miminis and Paige [1994]. Here we will simply give an example to demonstrate how the above transformation facilitates the assignment of two eigenvalues. Our example uses a system with $n = 9$, $m = 2$, $k = 5$, $n_1 = n_2 = n_3 = n_4 = 2$, and $n_5 = 1$; thus, $r = 4$ and $q = 2$. The transformation Q is computed as a product of four orthogonal matrices $Q = P_1P_2P_3P_4$ as follows: the matrix P_1 is a product of

$q - 1$ rotations that are applied on the right side of A_1, eliminating all but the first diagonal element of $A_{r,r-1}$ into their right-hand neighbors. P_1^T is then applied to the left of (A_1P_1, B_1), introducing a nonzero element to the left of the top diagonal element of $A_{r,r-1}$ and one nonzero element to the left of each of the next $q - 2$ diagonal elements of $A_{r-1,r-2}$. Note that, although in theory the whole of $Q^T = P_4^T P_3^T P_2^T P_1^T$ is applied to the left of B_1, in practice, due to the form of B_1, only part of Q affects it. In view of that, B_1 will appear, in our example, only when it is affected. The effect of transforming A_1 to $P_1^T A_1 P_1$ may be represented diagrammatically as follows:



where + denotes a newly introduced nonzero element. P_2 is now formed as a product of either $2(q - 1)$ rotations or $q - 1$ Householder transformations, which eliminate the newly introduced nonzero elements along with the associated diagonal elements, into their right-hand neighbors. P_2^T is then applied to the left of $(P_1^T A_1 P_1 P_2, P_1^T B_1)$, introducing new nonzero elements as follows:

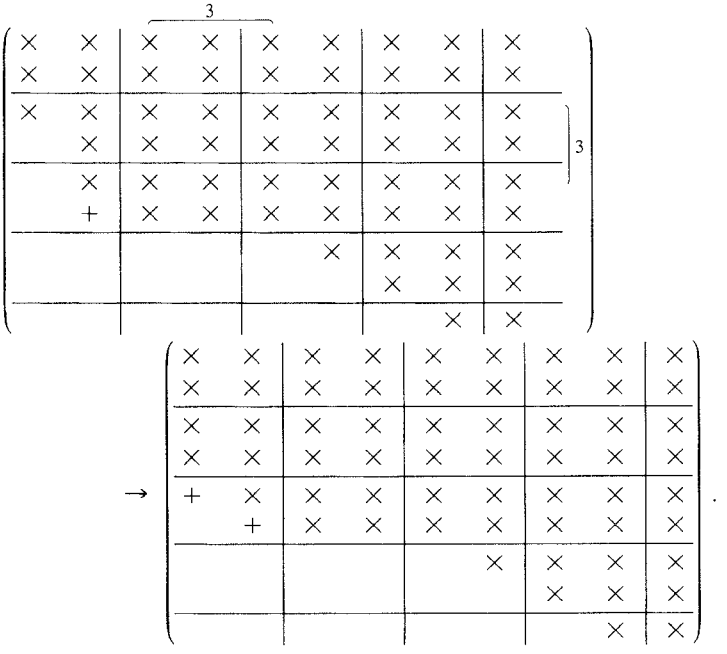




where Householder transformations were used in our example. P_3 now consists of either two rotations or one Householder transformation that eliminates elements $n_1 + (r - 4)q + 1$ and $n_1 + (r - 4)q + 2$ into element $n_1 + (r - 4)q + 3$ of the $n_1 + (r - 2)q + 1$ row of the real matrix

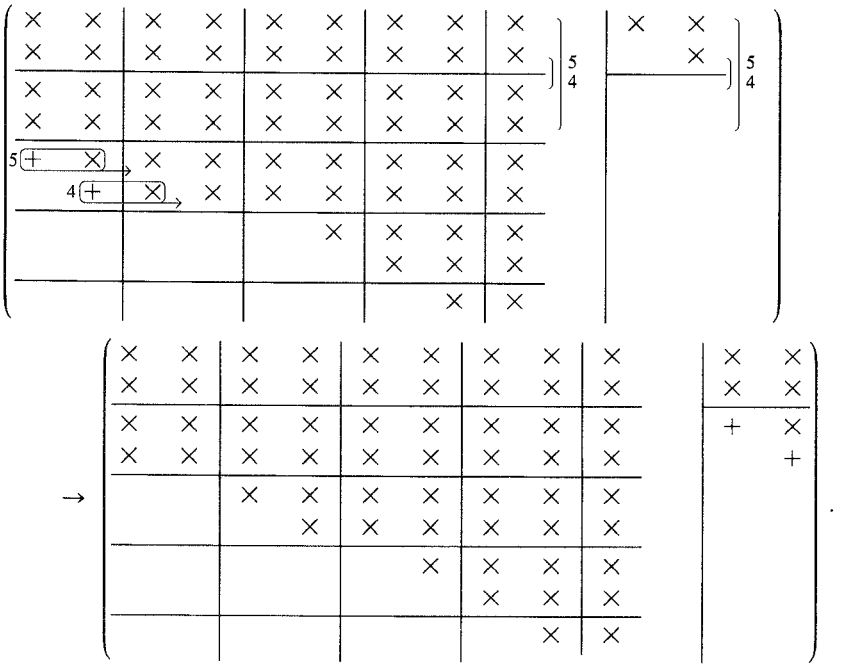
$$(A_1 - \lambda_1 I)(A_1 - \lambda_2 I)P_1P_2.$$

This is the only part of the eigenvalue assignment where λ_1 and λ_2 explicitly take part in the computation. Note that, even if λ_1, λ_2 is a complex conjugate pair, the matrix $(A_1 - \lambda_1 I)(A_1 - \lambda_2 I)$ is still real, and thus, the computation will continue in real arithmetic. Also note that only one row of $(A_1 - \lambda_1 I)(A_1 - \lambda_2 I)$ needs to be computed. Once P_3 is known, it is applied as $P_3^T(P_2^TP_1^TA_1P_1P_2P_3, P_2^TP_1^TB_1)$, with the following effect:



Summarizing, we see that each rotation of P_1 and each Householder transformation of P_2 eliminate elements from a distinct row of the transformed A_1. They start with row $n_1 + (r - 1)q$ going up to row $n_1 + (r - 3)q + 1$. P_3, however, eliminates no elements of $P_2^TP_1^TA_1P_1P_2$; it simply combines the

appropriate columns. Next we compute P_4 as a product of $n_1 + (r-4)q$ Householder transformations (two in our example) or twice as many rotations to eliminate the first two nonzero elements of rows $n_1 + (r-3)q + 2 : -1 : q + 3$ into the third element of the same row. P_4 is applied as $P_4^T(P_3^T P_2^T P_1^T A_1 P_1 P_2 P_3 P_4, P_3^T P_2^T P_1^T B_1)$, with the following effect:



It is important to point out that each Householder transformation of P_4 should first be applied on both sides of A_1 before the next transformation is computed. It is also worth noting that of all the above transformations only the last n_1 Householder transformations (or $2n_1$ rotations) affect B_1.

In the general case, the application of Q on $A_1 - B_1F_1$ will have the following effect:

$$Q^T(A_1 - B_1F_1)Q \equiv \left(\begin{array}{c|c} Q^TA_1Q(1:2,1:2) & Q^TA_1Q(1:2,3:n) \\ \hline Q^TA_1Q(3:n_1+2,1:2) & Q^TA_1Q(3:n,3:n) \\ \hline O & \end{array}\right) - \left(\begin{array}{c} Q^TB_1(1:2,:) \\ \hline Q^TB_1(3:n_1+2,:) \\ \hline O \end{array}\right)(F_1Q(:,1:2), F_1Q(:,3:n)) \tag{7}$$

$$= \left(\begin{array}{c|c} \begin{array}{c} Q^TA_1Q(1:2,1:2) \\ -\left[Q^TB_1(1:2,:)\right]\left[F_1Q(:,1:2)\right] \end{array} & \star \\ \hline \begin{array}{c} Q^TA_1Q(3:n_1+2,1:2) \\ -\left[Q^TB_1(3:n_1+2,:)\right]\left[F_1Q(:,1:2)\right] \end{array} & \\ \hline O & A_2 - B_2F_2 \end{array}\right),$$

where the ★ at the top right-hand corner of (7) represents a submatrix that does not affect the eigenvalue assignment. It is shown in Miminis and Paige [1994] that if Q is computed as described above and if $F_1Q(:,1:2)$ in (7) is computed so that

$$\left[Q^TB_1(3:n_1+2,:)\right]\left[F_1Q(:,1:2)\right] = Q^TA_1Q(3:n_1+2,1:2) \tag{8}$$

then the 2×2 submatrix

$$Q^TA_1Q(1:2,1:2) - \left[Q^TB_1(1:2,:)\right]\left[F_1Q(:,1:2)\right]$$

at the top left-hand corner of (7) has eigenvalues λ_1, λ_2. Hence, a pair of eigenvalues has been assigned, and if we set $A_2 \equiv Q^TA_1Q(3:n,3:n)$, $B_2 \equiv \begin{pmatrix} B_{22} \\ O \end{pmatrix}$ with $B_{22} \equiv Q^TB_1(3:n_1+2,:)$, and $F_2 \equiv F_1Q(:,3:n)$, we may continue the assignments with $A_2 - B_2F_2$.

It is shown in Miminis and Paige [1994] that the above procedure can only treat the case $r > 2$. When $r = 2$, we need slightly different algorithms for $n_1 = n_2$ and $n_1 = n_2 + 1$. Let us initially consider $n_1 = n_2$. Here the orthogonal transformation that facilitates the eigenvalue assignment is simply $Q = P_1$, where P_1 is computed exactly as in the $r > 2$ case, that is, as a product of $q - 1$ rotations (note that currently $A_{r,r-1} \equiv A_{21}$). The use of an example facilitates understanding; we therefore consider the system $n = 4$, $m = 2$, $k = 2$, $n_1 = n_2 = 2$, $q = 2$. Then

$$Q^TA_1Q - Q^TB_1F_1Q = \left(\begin{array}{cc|cc} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \hline \times & \times & \times & \times \\ & & \times & \times \end{array}\right) - \left(\begin{array}{cc|cc} \times & \times & & \\ 0 & \times & & \\ \hline & & \times & \times \\ & & 0 & \end{array}\right)\left(\begin{array}{cc|cc} \star & \star & \star & \star \\ \star & \star & \star & \star \end{array}\right)$$

$$= \left(\begin{array}{c|c} Q^TA_1Q(1:2,1:2) & Q^TA_1Q(1:2,3:n) \\ \hline Q^TA_1Q(3:q+1,1:2) & \\ \hline O & A_2 \end{array}\right) \tag{9}$$

$$- \left(\begin{array}{c|c} Q^TB_1(1:2,1) & Q^TB_1(1:2,2:m) \\ \hline & B_2 \end{array}\right)$$

$$\times \left(\begin{array}{c|c} F_1Q(1,1:2) & F_1Q(1,3:n) \\ \hline F_1Q(2:m,1:2) & F_2 \end{array}\right)$$

where $A_2 \equiv Q^T A_1 Q(3:n, 3:n)$, $B_2 \equiv \begin{pmatrix} B_{22} \\ O \end{pmatrix}$ with $B_{22} \equiv Q^T B_1(3:q+1, 2:m)$, and $F_2 \equiv F_1 Q(2:m, 3:n)$. Since B_{22} is nonsingular and upper triangular, $F_1 Q(2:m, 1:2)$ can easily be computed so that

$$B_{22}[F_1 Q(2:m, 1:2)] = Q^T A_1 Q(3:q+1, 1:2).$$

Having computed $F_1 Q(2:m, 1:2)$, we can write the 2×2 top left block of $Q^T A_1 Q - Q^T B_1 F_1 Q$ as

$$\begin{aligned} &\left[Q^T A_1 Q(1:2, 1:2) - \left[Q^T B_1(1:2, 2:m)\right]\left[F_1 Q(2:m, 1:2)\right]\right] \\ &\qquad - \left[Q^T B_1(1:2, 1)\right]\left[F_1 Q(1, 1:2)\right] \\ &\equiv \begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix} - \begin{pmatrix} \beta_1 \\ 0 \end{pmatrix} f_1^T, \end{aligned} \tag{10}$$

where $\beta_1 \neq 0$, and α_{21} can be proved to be nonzero (see Miminis and Paige [1994]). The above is a controllable two-dimensional single-input system. We may then assign a pair of eigenvalues λ_1, λ_2 by computing $f_1^T \equiv F_1 Q(1, 1:2)$ in (10) as

$$f_1^T = \frac{\alpha_{11} + \alpha_{22} - (\lambda_1 + \lambda_2),\ \alpha_{12} + (\alpha_{22} - \lambda_1)(\alpha_{22} - \lambda_2)/\alpha_{21}}{\beta_1}.$$

Furthermore, we may observe that $F_1 Q(1, 3:n)$, which is part of the $(1,2)$ block of (9), does not affect the assignment. Therefore, it can be chosen so that the first row of the $(1,2)$ block, given by

$$Q^T A_1 Q(1:2, 3:n) - \left[Q^T B_1(1:2, 1)\right]\left[F_1 Q(1, 3:n)\right] - \left[Q^T B_1(1:2, 2:m)\right] F_2, \tag{11}$$

may take any desirable value. This can always be accomplished since $e_1^T[Q^T B_1(1:2, 1)] \neq 0$, and thus, Eq. (11) may be solved with respect to $F_1 Q(1, 3:n)$ once F_2 is known. In this particular implementation, we set it to zero, which decreases the magnitude of F. We then continue the eigenvalue assignments with the controllable system (B_2, A_2) by computing F_2.

The case $r = 2$, $n_1 = n_2 + 1$, and no real eigenvalue to assign can be treated in a similar way. Here $Q = P_1 P_2$, with P_1 being a product of $q - 1$ rotations as in the $n_1 = n_2$ case, and P_2 a permutation matrix that will interchange columns one and three of $A_1 - B_1 F_1$ (see Miminis and Paige [1994] for justification). We have seen up to now that B_{11} in B_1 is upper triangular and that Q^T applied to the left of B_1 maintains this structure in B_{22}. In the current case, however, the upper triangular form of B_{11} is slightly perturbed by P_2^T. To see this, consider the example $n_1 = 3$, $q = n_2 = 2$. With

P_1 being only one rotation, we have

$$B_1 = \begin{pmatrix} + & \times & \times \\ & + & \times \\ & & + \\ & O & \end{pmatrix} \Rightarrow P_1^T B_1 = \begin{pmatrix} + & \times & \times & \\ & + & \times & \\ & & \times & \}1 \\ & & + & \\ & O & & \end{pmatrix}$$

$$\Rightarrow P_2^T P_1^T B_1 = \begin{pmatrix} & & \times & \\ & + & \times & \}2 \\ + & \times & \times & \\ & & + & \\ & O & & \end{pmatrix},$$

where + represents a strictly nonzero element. At this point we may either continue with the above form, or apply a rotation P to the right of $P_2^T P_1^T B_1$ to eliminate the (3, 1)-element into its right-hand neighbor as

$$P_2^T P_1^T BP = \left(\begin{array}{c|cc} 0 & & \times \\ + & \times & \times \\ \hline & + & \times \\ & & + \\ & O & \end{array}\right).$$

The latter is the approach that has been implemented in the procedure **MEVAS**. The rest is similar to the $r = 2$, $n_1 = n_2$ case except that this time the resulting 2×2 single-input EAP has the form

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{pmatrix} - \begin{pmatrix} 0 \\ \beta_1 \end{pmatrix} f_1^T$$

with $\alpha_{12} \neq 0$ instead. For more details, see Miminis and Paige [1994].

It is apparent that, when the case $n_1 = n_2 + 1$ occurs and a real eigenvalue is available, we have the following choices: either assign the real eigenvalue immediately in a single step (similar to that in (5) and (6)), or continue with a double step. Different choices will make the algorithm finish in different ways. Although this is theoretically fine, it can be a problem when the algorithm is to be implemented, since its end should be known a priori so appropriate actions may be taken. Here we will describe the end of the algorithm when the following procedure is adopted. First, we assign "immediately" s eigenvalues in (5) and (6), and continue with a system that has $n \leftarrow n - s$ states. If n is even, we only perform double steps; if n is odd, we allow only one single step when the case $n_1 = n_2 + 1$ occurs for the first time. At this step a real eigenvalue will be assigned "immediately" (in **MEVAS**, the nth eigenvalue), and we will continue with a system that has an even number of states. The remaining eigenvalues will be assigned in double steps. Next we show that, working in this way, the form of the final system can be determined by the initial value of k (in (4)) only. At each double step, the number of states of the resulting system decreases by two. Since n will

eventually become even, the algorithm will always finish with a 2×2 system. To find the form of the final 2×2 system (let us name it $\mathcal{S}$), we can make the following observations for k. At each double step, the block rows r and $r - 1$ of the current matrix A_1 decrease by one row each. When these blocks consist of only one row each, they simply vanish. When that happens, k decreases by two. If k is odd, it will eventually become one. In this case the number of states of $\mathcal{S}$ equals the number of its inputs, and the continuation is trivial. For example, solve for $\hat{F}$ in

$$\hat{A} - \hat{B}\hat{F} = \begin{pmatrix} \zeta_1 & \eta \\ -\eta & \zeta_2 \end{pmatrix},$$

where $\zeta_1 + \iota\eta$ and $\zeta_2 - \iota\eta$ are the last two eigenvalues, with $\zeta_1 = \zeta_2$ when $\eta \neq 0$. If k starts even, it will eventually become two. In this case $\mathcal{S}$ will be a 2×2 single-input system of the form (10) and can be treated in the same way.

When this part of the algorithm finishes, $\tilde{F} = \tilde{P}^T F_1 \tilde{Q}$ has been computed. Here $\tilde{Q}$ represents all of the orthogonal transformations Q, used at each double step, and $\tilde{P}$ represents all of those orthogonal transformations P_2 used when the case $r = 2$ and $n_1 = n_2 + 1$ occurs. Note that neither $\tilde{Q}$ nor $\tilde{P}$ are the products of all the relevant transformations Q or P, respectively. During the eigenvalue assignment, each Q and each P were not applied to the whole of F_1, but only to the relevant submatrix of F_1. This should be taken into consideration when the transformations are applied back in order to compute F_1 from $\tilde{F}$. In view of this, the use of $\tilde{P}^T F_1 \tilde{Q}$ above is an "abuse" of notation, and it simply denotes the application of individual transformations P^T, Q to the relevant submatrices of F_1. In our software this part is implemented by the subroutine **MEVAS**. Having now computed F_1, we can compute the original F as

$$F = UF_1T^T, \tag{12}$$

using the transformations in (4). In our software this part is implemented by the subroutine **BKTRN**.

2.2 Experimenting with the Algorithm

The aim of our algorithm is to compute an F such that $\lambda(A - BF)$ is a specified set of self-conjugate numbers. Here $\lambda(\cdot)$ represents the set of eigenvalues of a matrix. In view of this, we have

$$(A - BF)X = XJ, \tag{13}$$

where J and X are the Jordan canonical form and the matrix of the eigenvectors of $A - BF$, respectively. It is well known that, in the multiinput case ($m > 1$), F is not unique. Furthermore, it has been shown in Kautsky et al. [1985] that the nonuniqueness of F is due to the fact that any vector $x_i \in \mathcal{N}[V^T(A - \lambda_i I)]$, $i = 1:n$, may qualify as an eigenvector of $A - BF$ corresponding to eigenvalue λ_i, where $\mathcal{N}(\cdot)$ represents the null space of a matrix and where V is an orthonormal matrix that spans $\mathcal{N}(B^T)$. Although

our algorithm is not directly concerned with the assignment of a particular eigenvector, it obviously assigns some eigenvector implicitly.

Testing the effectiveness of an algorithm for the EAP may be twofold. That is, one may check the accuracy of the computed F as well as the accuracy of the computed eigenvalues of $A - BF$ against those specified for assignment.

The accuracy of the computed F depends on the stability of the algorithm as well as the conditioning of the EAP. In Miminis and Paige [1988], the numerical stability of the single-step algorithm for the same problem was proven. Although the stability of the double-step algorithm presented here and in Miminis and Paige [1994] has not been proved as yet, there is strong evidence that the algorithm is numerically stable (we do, however, intend to prove this formally in the near future). The condition number of the EAP has not been formally found as yet; there has been strong experimental evidence, however, as well as theoretical evidence (for the latter, see Demmel [1987]), that at least two factors affect it. The first factor is the relative "distance" (δ) of the given system from the nearest uncontrollable system (for a definition, see Paige [1981]), and the second is $\eta = min|\mu - \lambda|$, with $\lambda \in \lambda(A)$ and $\mu \in \lambda(A - BF)$. The condition number appears to depend on η/δ. The factor η is a good estimation of how "far" the eigenvalues of A are from those to be assigned. In order to test the effectiveness of the algorithm with respect to the accuracy of F, we ran numerous experiments with systems of size up to $n = 50$ and varying m (including $m = 1$ when $n = 50$). Each run included the computation of F in double precision (F_d) and in single precision (F_s). Since F_d is at least twice as accurate as F_s, the quantity $\|F_d - F_s\|/\|F_d\|$ was used as an acceptable estimation of the accuracy of F_s. In each run, where δ and η were of reasonable magnitudes, the algorithm gave accurate results to machine precision. We have also tested the algorithm with systems that are nearly uncontrollable (small δ); as expected, the algorithm produced inaccurate solutions. We observed, however, that nearly uncontrollable systems manifested their existence by producing ill-conditioned systems of equations in (8). They may therefore be identified, and the process may be aborted before inaccurate columns of F contaminate the already accurately computed columns. In this way, although the algorithm does not assign all of the eigenvalues, those that are already assigned are assigned accurately. This has been implemented in **MEVAS**, which in this case produces an F accurate to machine precision (the level of accuracy depends on the tolerance used; see below) so that some of the eigenvalues of $A - BF$ are among those specified, with the remaining eigenvalues being any. At this point it is worth mentioning that in the single-input ($m = 1$) case, systems (8) are scalar, and scalar equations are never ill conditioned. In this case an ill-conditioned EAP may appear via (8) as a scalar equation with a zero (less than the tolerance used) left-hand side. This obviously can be detected and avoided. Two of the examples in our software demonstrate this. Note that in the case of an ill-conditioned EAP the number of eigenvalues assigned depends on the tolerance used (see TOL in Section 4.2). A small tolerance will allow more eigenvalues to be assigned than a larger tolerance. On the other hand, as more eigenvalues are assigned, the columns of F tend to become more

inaccurate. The value of the tolerance is fully controlled by the user (see Section 4.2), who therefore may control the above process.

Finally, we tested the algorithm with problems where η was relatively large. Our results indicated that the effect of the magnitude of η on F was twofold. First, we observed that a large η affected the accuracy of F, which was expected. An η, however, not large enough to affect seriously the accuracy of F, may still cause problems by being responsible for an F that produces a matrix $A - BF$ with an ill-conditioned eigenproblem.

The following analysis supports our claim. Let J in (13) be diagonal; then, from the Bauer–Fike theorem (e.g., see Golub and Van Loan [1989]), we have

$$\|F\|\mathscr{K}(X) \geq \frac{\eta}{\|B\|}, \tag{14}$$

where BF may be considered as a perturbation to A and where $\mathscr{K}(X) = \|X\|\|X^{-1}\|$. Note that the Bauer–Fike theorem is well known for showing that $\mathscr{K}(X)$ is the condition number of the eigenproblem of a matrix ($A - BF$ in our case) with eigenvectors being the columns of X. From (14) we may conclude that, if η is large, either $\|F\|$ is large, or $\mathscr{K}(X)$ is large (in the latter case, $A - BF$ has an ill-conditioned eigenproblem). In Kautsky et al. [1985], however, it is shown that

$$\|F\| \leq \frac{\|A\| + \|J\|\mathscr{K}(X)}{\sigma_{min}(B)}, \tag{15}$$

with $\sigma_{min}(B)$ being the minimum singular value of B. From (15) we see that, if $\|F\|$ is large, so is $\mathscr{K}(X)$. Now combining (14) and (15), we see that, if η is large enough, $A - BF$ has an ill-conditioned eigenproblem.

3. COMPUTATIONAL COST OF THE ALGORITHM

In this section we analyze the computational cost of the algorithm presented in Section 2. In Section 3.1 we give the time complexity of the algorithm in terms of an order on the number of multiplications. In Section 3.2 an analysis on the storage requirements is performed. This analysis produces very strict bounds since it is essential for **MEVAS**.

3.1 Time

The algorithm in Section 2 begins with a transformation that leads to form (4). Procedure **MEVAS**, however, assumes that the system is already in form (4) and proceeds with the assignments. Although form (4) is not computed by **MEVAS**, it is worth mentioning that it takes

$$O\left(\tfrac{5}{3}n^3 + mn^2\right)$$

multiplications to be produced when the QR decomposition is used (instead of the SVD) to compute the ranks. Next we consider the part of the algorithm that assigns the eigenvalues. At each double assignment, we need to compute and apply an orthogonal matrix Q. To analyze this consider a system in form

(4), where $r = k$ and $n_1 = n_2 = \cdots = n_k = q$; thus, $rq = t$ is the size of the matrix A_1. Obviously, any other system with $r < k$ will involve less computations than the above. Thus, if we assume that every assignment takes place on a system with $r = k$, we will produce an upper bound on the number of multiplications of the worst case of our algorithm. We know that $Q = P_1P_2P_3P_4$. P_1 consists of $q - 1$ rotations; each rotation, say, R_i, applied on a vector involves 4 multiplications. R_i is applied on $rq - i + 1$ vectors, whereas R_i^T is applied on $2q + i$ vectors with $i = 1 : q - 1$. Thus, to form $R_i^T A R_i$ we need $4((r + 2)q + 1)$ multiplications, whereas to form $P_1^T A P_1$ we need

$$4(q - 1)[(r + 2)q + 1] \tag{16}$$

multiplications. P_2P_3 now consists of q Householder transformations, say, H_i. Each H_i applied on an ℓ-dimensional vector involves 2ℓ multiplications. Note that in our algorithm we always have $\ell = 3$. H_i is applied on $(r - 1)q + 2 - i$ three-dimensional vectors, while H_i^T is applied on $3q + i$ vectors. Thus, to form $P_3^T P_2^T (P_1^T A P_1) P_2 P_3$ we need

$$6q[(r + 2)q + 2] \tag{17}$$

multiplications. P_4 finally consists of Householder transformations H_i that start from row $(r - 2)q + 2$ and go up to row $q + 3$; hence, P_4 consists of $(r - 3)q$ Householder transformations. For each of the first $(r - 4)q$ of these Householder transformations, H_i is applied on $(r - 2)q + 3 - i$ vectors, whereas H_i^T is applied on $4q + i$ vectors. Hence, to apply both H_i and H_i^T, $i = 1 : (r - 4)q$, we need

$$6(r - 4)q((r + 2)q + 3) \tag{18}$$

multiplications. For the remaining q Householder transformations, H_i is applied on $2q + 3 - i$ vectors, while H_i^T is applied on $(r + 1)q$ vectors (since B_{11} is included this time). Hence, we need a total number of

$$(r + 13)q^2 + 17q \tag{19}$$

multiplications. Summing up (16)–(19), in order to form $Q^T A Q$ we need

$$(6r^2 - r + 81)q^2 + (14r - 79)q - 1 = O\big(6(rq)^2\big) \equiv O(6t^2)$$

multiplications. Now assume for simplicity that n is even. The number of multiplications we then need to perform $n/2$ such double steps is given by

$$6 \sum_{t = 2.2.n} t^2 = 6 \sum_{t=1}^{\frac{n}{2}} (2t)^2 = O(n^3). \tag{20}$$

In each double step, we need to solve two upper triangular systems of equations of the type (8). The solution of an $\ell \times \ell$ upper triangular system of equations requires $O(\ell^2/2)$ multiplications. Here we solve n such systems

of order starting from m and decreasing down to 1. Thus, it is safe to say that the number of multiplications for this part is

$$O\left(\frac{nm^2}{2}\right). \tag{21}$$

Finally, we need to compute $F = U\tilde{P}\tilde{F}\tilde{Q}^T T^T$. Here, however, we do not consider transformations U, T that lead to form (4). Neither will we consider the effect of $\tilde{P}$, which is negligible compared with that of $\tilde{Q}^T$. To simplify the process further, we assume that $\tilde{Q}^T$ is applied to the whole of $\tilde{F}$, instead of just to the appropriate submatrix of $\tilde{F}$. $\tilde{Q}$ consists of $n/2$ transformations of the type Q that is discussed in Section 2. Each Q consists of $q - 1$ rotations and $(r - 2)q$ Householder transformations. Each of these transformations is applied on m vectors (m rows of $\tilde{F}$). Hence, it needs

$$4m(q - 1) + 6m(r - 2)q \tag{22}$$

multiplications. But $r \leq k$, and $q \leq m$; hence, (22) is bounded by

$$4m(m - 1) + 6m(k - 2)m. \tag{23}$$

Multiplying now (23) by $n/2$ and assuming that $(k + 1)m = n$ (in the average case, $n = mk + n \bmod m$), we find that the application of $\tilde{Q}^T$ on $\tilde{F}$ requires

$$O(3n^2m - 7nm^2) \tag{24}$$

multiplications. From (20), (21), and (24), we get that the algorithm needs

$$O\left(n^3 + 3n^2m - \frac{13}{2}nm^2\right)$$

multiplications. The above assumes its maximum value when $m = (3/13)n$. This value is $O(1.506n^3)$.

3.2 Storage

The storage requirements of the algorithm described in Section 2 are dominated by the number of Householder transformations and the number of Givens rotations. Here we derive upper bounds on these numbers. Without loss of generality, assume that $rank(B) = m$. Let r, q, and n_i with $i = 1:k$ be defined as in Section 2. Also recall that rotations and Householder transformations arise in the following types of assignments:

Type 1. $r > 2$. This assignment requires $q - 1$ rotations (one dummy rotation if $q = 1$) for P_1, $q - 1$ Householder transformations for P_2, one Householder transformation for P_3, and $n_1 + (r - 4)q$ Householder transformations for P_4.

Type 2a. $r = 2$ *and* $n_1 = n_2$. This assignment requires $q - 1$ rotations for P_1 and one dummy Householder transformation for P_2.

Type 2b. $r = 2$ *and* $n_1 = n_2 + 1$. This assignment requires $q - 1$ rotations for P_1, one trivial Householder transformation for P_2, and one rotation for P.

The total number of assignments of each type (hence, total number of rotations and Householder transformations) can be determined by observing the change in the indices n_i, $i = 1:k$, as the assignment proceeds. To this end let α, β be nonnegative integers such that $n = \alpha m + \beta$, $0 \le \beta < m$, and consider the following four cases. Initially, we assume that none of the subdiagonal blocks of A_1 in (4) are rank deficient:

Case 1. $\alpha > 2$ *and* α *even.* The assignments proceed as follows (see the example below): for each $q = m:-1:\beta+1$, we have $(\alpha-2)/2$ double steps of Type 1 and one double step of Type 2. For each $q = \beta:-1:2$, we have $\alpha/2$ double steps of Type 1. Finally, for $q = 1$ we need $\alpha/2$ double steps of Type 1. As an example consider the case $n = 26$, $m = 4$; hence, $\alpha = 6$ and $\beta = 2$. Then r, q, and the indices n_i, $i = 1:7$, will change as follows:

$n_i, i = 1:7$							r	q		
4	4	4	4	4	4	2	6	4	$\frac{(\alpha-2)}{2}$ of $r > 2$	$q = 4 = m,$
4	4	4	4	3	3	2	4	4		
4	4	3	3	3	3	2	2	4	1 of $r = 2$	
3	3	3	3	3	3	2	6	3	$\frac{\alpha-2}{2}$ of $r > 2$	$q = 3,$
3	3	3	3	2	2	2	4	3		
3	3	2	2	2	2	2	2	3	1 of $r > 2$	
2	2	2	2	2	2	2	7	2	$\frac{\alpha}{2}$ of $r > 2$	$q = 2 = \beta,$
2	2	2	2	2	1	1	5	2		
2	2	2	1	1	1	1	3	2		
2	1	1	1	1	1	1	7	1	$\frac{\alpha}{2}$ of $r > 2$	$q = 1.$
2	1	1	1	1	0	0	5	1		
2	1	1	0	0	0	0	3	1		
2	0	0	0	0	0	0	last		assignments	

Let g_1 be the number of Givens rotations associated with Case 1. Then, in view of the above,

$$g_1 = \sum_{q=\beta+1}^{m} \left\{ \frac{\alpha-2}{2}(q-1) + (q-1) \right\} + \sum_{q=2}^{\beta} \left\{ \frac{\alpha}{2}(q-1) \right\} + \frac{\alpha}{2}(1)$$
$$= \frac{\alpha}{2}\left[1 + \frac{m(m-1)}{2} \right].$$

Anticipating the results of the case $\alpha = 2$, we note here that $g_1 < \alpha/2[1 + (m(m-1))/2] + (m-\beta)$.

The summation of the Householder transformations is a little more complicated because of the presence of $n_1 + (r-4)q$ transformations associated with each Type 1 assignment. Note that r takes on a sequence of values at

each q. Denoting this sequence by R_q and letting h_1 denote the number of Householder transformations associated with Case 1, we find that

$$h_1 = \sum_{q=\beta+1}^{m} \left\{ \frac{\alpha-2}{2}(q) + \sum_{r\in R_q} [n_1 + (r-4)q] + 1 \right\} + \sum_{q=2}^{\beta} \left\{ \frac{\alpha}{2}(q) + \sum_{r\in R_q} [n_1 + (r-4)q] \right\} + \left\{ \frac{\alpha}{2}(1) + \sum_{r\in R_1} [n_1 + (r-4)1] \right\}. \tag{25}$$

Observe that

(1) for $\beta + 1 \le q \le m$, $n_1 = q$ and $R_q = \{\alpha, \alpha - 2, \ldots, 2\}$; thus,

$$\sum_{r\in R_q} [n_1 + (r-4)q] = (1 + 3 + \cdots + \alpha - 3)q;$$

(2) for $2 \le q \le \beta$, $n_1 = q$ and $R_q = \{\alpha + 1, \alpha - 1, \ldots, 3\}$; thus,

$$\sum_{r\in R_q} [n_1 + (r-4)q] = (2 + 4 + \cdots + \alpha - 2)q;$$

(3) for $q = 1$, $n_1 = 2$ and $R_1 = \{\alpha + 1, \alpha - 1, \ldots, 3\}$; thus,

$$\sum_{r\in R_1} [n_1 + (r-4)1] = (1 + 3 + \cdots + \alpha - 1).$$

Putting these relations into Eq. (25) yields

$$\begin{aligned} h_1 &= \sum_{q=\beta+1}^{m} \left\{ \frac{\alpha-2}{2}q + (1 + 3 + \cdots + \alpha - 3)q + 1 \right\} \\ &\quad + \sum_{q=2}^{\beta} \left\{ \frac{\alpha}{2}q + (2 + 4 + \cdots + \alpha - 2)q \right\} + \left\{ \frac{\alpha}{2} + (1 + 3 + \cdots + \alpha - 1) \right\} \\ &= \frac{\alpha}{2}\left[1 + \frac{\beta(\beta+1)}{2} + \frac{(\alpha-2)}{2}\frac{m(m+1)}{2} \right] + m - \beta \\ &\le \frac{\alpha}{2}\left[1 + \frac{\beta(\beta+1)}{2} + \frac{(\alpha-2)}{2}\frac{m(m+1)}{2} \right] + m. \end{aligned}$$

Let g_i and h_i denote, respectively, the number of Givens rotations and Householder reflectors associated with the case $i = 2, 3, 4$. By analyses similar to the above, we find the following:

Case 2. $\alpha = 2$.

$$g_2 = 1 + \frac{m(m-1)}{2} + (m - \beta),$$
$$h_2 = \frac{\beta(\beta+1)}{2} + m - \frac{\beta}{2} \leq \frac{\beta(\beta+1)}{2} + m.$$

Now, since $\alpha/2 = 1$ and $(\alpha - 2)/2 = 0$ we can write

$$g_2 = \frac{\alpha}{2}\left[1 + \frac{m(m-1)}{2}\right] + (m - \beta),$$
$$h_2 \leq \frac{\alpha}{2}\left[1 + \frac{\beta(\beta+1)}{2} + \frac{\alpha-2}{2}\frac{m(m+1)}{2}\right] + m.$$

Case 3. $\alpha > 2$ *and* α *odd.*

$$g_3 = \frac{\alpha-1}{2}\left[1 + \frac{m(m-1)}{2}\right] + \frac{\beta(\beta-1)}{2}$$
$$\leq \frac{\alpha-1}{2}\left[1 + \frac{m(m-1)}{2}\right] + \frac{\beta(\beta+1)}{2},$$
$$h_3 = \frac{\alpha-1}{2}\left[\frac{(m-\beta)}{2} + \frac{\beta(\beta+1)}{2} + \frac{\alpha-1}{2}\frac{m(m+1)}{2}\right] + \beta - 1$$
$$< \frac{\alpha-1}{2}\left[(m-\beta) + \frac{\beta(\beta+1)}{2} + \frac{\alpha-1}{2}\frac{m(m+1)}{2}\right] + \beta.$$

Case 4. $\alpha = 1$.

$$g_4 = \frac{\beta(\beta+1)}{2} - 1 < \frac{\beta(\beta+1)}{2},$$
$$h_4 = \beta - 1 < \beta.$$

Since $(\alpha - 1)/2 = 0$, we can write

$$g_4 < \frac{\alpha-1}{2}\left[1 + \frac{m(m-1)}{2}\right] + \frac{\beta(\beta+1)}{2},$$
$$h_4 < \frac{\alpha-1}{2}\left[(m-\beta) + \frac{\beta(\beta+1)}{2} + \frac{\alpha-1}{2}\frac{m(m+1)}{2}\right] + \beta.$$

In summary, we have the following bound g on the number of Givens rotations:

$$g \leq \begin{cases} \dfrac{\alpha}{2}\left[1 + \dfrac{m(m-1)}{2}\right] + (m - \beta), & \alpha \text{ even}, \\ \dfrac{\alpha - 1}{2}\left[1 + \dfrac{m(m-1)}{2}\right] + \dfrac{\beta(\beta+1)}{2}, & \alpha \text{ odd}; \end{cases}$$

and the following bound h on the number of Householder transformations:

$$h \leq \begin{cases} \dfrac{\alpha}{2}\left[1 + \dfrac{\beta(\beta+1)}{2} + \dfrac{\alpha - 2}{2}\dfrac{m(m+1)}{2}\right] + m, & \alpha \text{ even}, \\ \dfrac{\alpha - 1}{2}\left[(m - \beta) + \dfrac{\beta(\beta+1)}{2} + \dfrac{\alpha - 1}{2}\dfrac{m(m+1)}{2}\right] + \beta, & \alpha \text{ odd}. \end{cases}$$

Recall that the above bounds were derived under the assumption of no rank deficiencies in the subdiagonal blocks of the staircase form of A_1 in (4). As a final remark, note that the presence of such rank deficiencies will not increase the number of Householder transformations or rotations required, so that the above expressions are always valid upper bounds.

4. THE SOFTWARE

4.1 Overview

We provide implementations in both single and double precision, and follow the convention that names of single-precision routines begin with the letter "**S**" and double-precision routines with "**D**." For ease of exposition, we will refer only to the double-precision routines.

In addition to the subroutine **DMEVAS**, which implements the eigenvalue assignment algorithm described in Section 2, we also include subroutine **DSTAIR** to reduce matrices A and B to the form (4) required by **DMEVAS**, and subroutine **DBKTRN** to compute F from F_1 in (12). A typical application will call **DSTAIR**, **DMEVAS**, and **DBKTRN** in this order. We also provide a demonstration program illustrating the use of the software. This program also computes the eigenvalues of $A - BF$ to allow comparison with the eigenvalues specified for assignment.

The routines have been successfully run on a number of computers, including a CONVEX C-1 vector computer, an HP/720, a MIPS M-120, a DEC MICROVAX III, a Sun 4/260, and a ZENITH Z-note 325L notebook equipped with a 387 coprocessor. Although the above subroutines successfully run when $m = 1$ (single-input systems), we would recommend the use of **DSEVAS** instead [Miminis and Reid 1993], since this subroutine (also available in single precision) has been specifically produced for single-input systems and is faster than **DMEVAS** when $m = 1$. The algorithm has also been implemented as a **MATLAB** function, available from the authors upon request.[1]

4.2 The Subroutine **DMEVAS**

DMEVAS computes a real matrix F_1 so that the matrix $A_1 - B_1F_1$ has a specified set of self-conjugate eigenvalues. The system (B_1, A_1) is in upper staircase form (4). The subroutine heading and the argument declarations are as follows:

```
      SUBROUTINE DMEVAS (N, M, NCMPLX, GMAX, HMAX, A, LDA, B, LDB,
     &                   F, LDF, EIGS, KMAX, KSTAIR, INFO, IWORK,
     &                   RWORK, TOL, IWARN, IERR)
      INTEGER            N, M, NCMPLX, GMAX, HMAX, LDA, LDB, LDF,
     &                   IWARN, IERR, KMAX, KSTAIR(*), INFO(2),
     &                   IWORK(GMAX + HMAX + N / 2 + N)
      DOUBLE PRECISION   A(LDA, N), B(LDB, M), F(LDF, N), EIGS(N),
     &                   RWORK(2*GMAX + 3*HMAX + 3*N), TOL
```

The description of the *input arguments* is as follows:

—N: Row and column dimension of matrix A; row dimension of matrix B; length of vector of eigenvalues **EIGS**; $N \geq 1$.

—M: Column dimension of matrix B; row dimension of matrix F; $M \geq 1$.

—**NCMPLX**: Number of complex eigenvalues in **EIGS** (see also **EIGS** below). $0 \leq$ **NCMPLX** $\leq N$, and **NCMPLX** even.

—**GMAX**: Maximum number of Givens rotations to be used in the computation. A value of **GMAX** may be computed as follows: let $N = qM + r$, where $q = \lfloor N/M \rfloor$, $r = N \bmod M$. Set rsum $= r(r + 1)/2$ and Msum1 $= M(M - 1)/2$; then

$$\textbf{GMAX} = \begin{cases} \dfrac{q}{2}(1 + \text{Msum1}) + M - r, & \text{if } q \text{ is even,} \\ \dfrac{q - 1}{2}(1 + \text{Msum1}) + \text{rsum}, & \text{if } q \text{ is odd.} \end{cases}$$

—**HMAX**: Maximum number of Householder transformations to be used in the computation. A value of **HMAX** may be computed as follows: let q, r, rsum, and Msum1 be defined as for **GMAX** above. In addition, let Msum $= M(M + 1)/2$; then

$$\textbf{HMAX} = \begin{cases} \dfrac{q}{2}\left(\text{Msum}\dfrac{q - 2}{2} + \text{rsum} + 1\right) + M, & \text{if } q \text{ is even,} \\ \dfrac{q - 1}{2}\left(\text{Msum}\dfrac{q - 1}{2} + \text{rsum} + M - r\right) + r, & \text{if } q \text{ is odd.} \end{cases}$$

[1] It may also be obtained by connecting to *triton.cs.mun.ca* (134.153.1.19) by *anonymous ftp*; change directory to *pub/miminis*, and in binary mode get the file *PolePack.tar.Z*.

Less-stringent but simpler values for **GMAX** and **HMAX** may be provided by the following:

$$\mathbf{GMAX} = \frac{q}{2}(1 + \mathrm{Msum1}) + M\frac{r+2}{2},$$

$$\mathbf{HMAX} = \frac{q}{2}\left(\mathrm{Msum}\frac{q}{2} + \mathrm{rsum} + M - r\right) + M.$$

—A: The leading $N \times N$ part of this array must contain the matrix A_1 in staircase form (4). The array is overwritten during the process.

—**LDA**: Row dimension of array A, as declared in the calling program, with **LDA** $\geq N$.

—B: The leading $N \times M$ part of this array must contain the matrix B_1 in form (4). The array is overwritten during the process.

—**LDB**: Row dimension of array B, as declared in the calling program, with **LDB** $\geq N$.

—**LDF**: Row dimension of array F, as declared in the calling program, with **LDF** $\geq M$.

—**EIGS**: Vector of eigenvalues to be allocated. The complex eigenvalues (there are **NCMPLX** of them) must occur as conjugate pairs. They are stored in **EIGS**(1 : **NCMPLX**), and the real eigenvalues (there are $N -$ **NCMPLX** of them) are stored in **EIGS**(**NCMPLX** + 1 : N).

Since the real and imaginary parts of a complex number also determine its conjugate, only one real part and one imaginary part are stored for each pair of conjugates. These parts are stored in successive elements of **EIGS**, with the real parts having odd indices. To store, for example, the four complex eigenvalues (0.1, 0.2), (0.1, −0.2), (0.3, −0.4), (0.3, 0.4) and the two real eigenvalues 0.5, 0.6, **EIGS** may be initialized to 0.1, 0.2, 0.3, −0.4, 0.5, 0.6. Observe that, for odd $i <$ **NCMPLX**, **EIGS**(i) and **EIGS**(i + 1) are the real and imaginary parts, respectively, of a complex eigenvalue. The array may be rearranged during the process.

—**KMAX**: The number of blocks in the staircase form (4) of the system (B_1, A_1).

—**KSTAIR**: Array of dimension 1 + **KMAX**. Note that **KSTAIR**(1) = $rank(B_1)$ and **KSTAIR**(i) = $rank(A_{i,i-1})$, for $i = 2 :$ **KMAX**. **KSTAIR**(**KMAX** + 1) is used by the routine. The array is overwritten during the process.

The description of the *output arguments* is as follows:

—**NCMPLX**: Number of complex eigenvalues that were not allocated. Complex eigenvalues are always allocated as conjugate pairs, so **NCMPLX** will always be even.

—**F**: The leading $M \times N$ part of this array contains the matrix F_1, which is the solution to the $A_1 - B_1F_1$ EAP, with (B_1, A_1) in staircase form (4).

—**EIGS**: Vector of allocated eigenvalues followed by those that were not allocated, if any. The order of eigenvalues in **EIGS** may differ from the original insofar as the eigenvalue originally stored as **EIGS**(N) may be moved to **EIGS**(i), with $i \neq N$. Then the eigenvalues originally stored in **EIGS**($i : N - 1$) will be shifted to **EIGS**($i + 1 : N$), with no additional reordering. This can occur only if N is odd (and, hence, **EIGS**(N) is real), as is discussed at the end of Section 2.1. The index i is returned to the calling program in **INFO**(2) (see **INFO** below).

—**INFO**: INFO(1) returns the number of successfully allocated eigenvalues. INFO(2) returns index in EIGS, of the eigenvalue originally stored as EIGS(N) (see also EIGS above).

Regarding the remaining arguments, we have two work spaces **RWORK** and **IWORK**, and a tolerance **TOL** that is either a value specified by the user, or, if the user-specified value is less than the relative machine precision *eps*, **TOL** is reset to **TOL** = $(M + N)\|(B, A)\|_1 eps$. For details on *eps*, see routine **DLAMCH** in the **LAPACK**. Elements with a value less than **TOL** are considered zero. We also have a warning argument **IWARN** and an error argument **IERR**. Unless the routine detects an unusual case or an error, **IWARN** and **IERR** contain 0 on exit, respectively. If they return a nonzero value, it may be interpreted as follows:

—**IWARN** = 1: On entry, $M > N$.

—**IWARN** = 2: On entry, the sum of ranks of staircase blocks is not equal to N.

—**IWARN** = 3: On entry, conditions for **IWARN** = 1 and **IWARN** = 2 both exist.

—**IERR** < 0: **IERR** = $-j$ indicates a problem with the jth argument on entry. In particular,

IERR = −1; on entry, $N < 1$;

IERR = −2; on entry, $M < 1$;

IERR = −3; on entry, **NCMPLX** < 0, **NCMPLX** > N, or **NCMPLX** is odd;

IERR = −4; on entry, **GMAX** < 1;

IERR = −5; on entry, **HMAX** < 1;

IERR = −7; on entry, **LDA** < N;

IERR = −9; on entry, **LDB** < N;

IERR = −11; on entry, **LDF** < M.

—**IERR** = 1: Signifies an attempt to divide by zero (i.e., a magnitude less than **TOL**) or to solve a numerically singular system of equations. Normally, this implies that the system is uncontrollable to machine precision, but it can also occur as a result of errors in parameter passing.

—**IERR** = 2: During eigenvalue assignment, a rank deficiency is discovered in one of the staircase blocks, indicating that the system (B_1, A_1) is uncontrollable. Assignment of eigenvalues can proceed no further.

—**IERR** = 3: Signifies insufficient storage space for Givens rotations. The quantity **GMAX** needs to be increased.

—**IERR** = 4: Signifies insufficient storage space for Householder transformations. The quantity **HMAX** needs to be increased.

4.3 Storage of Orthogonal Transformations in **DMEVAS**

Each Householder transformation H used in the subroutine is computed to eliminate the first two elements of a three-dimensional vector into the third element. Thus, the transformation can be completely specified by a three-dimensional vector u (since $H = I - 2uu^T/u^Tu$; see Golub and Van Loan [1989]). In **DMEVAS** the vector u is computed so that its third element is normalized to unity. Since the value of $u(3)$ is known, it need not be stored, and $u(3)$ is used to store $u^Tu/2$ instead. The individual vectors are stored in columns of the $3 \times h$ array QH, where h represents the maximum number of Householder transformations that may be expected. Associated with each Householder transformation is an index indicating where the transformation is to be applied to a vector. This index is stored in the corresponding element of array HCOL. Thus, if the Householder transformation stored at QH(:, j) is to be applied to a vector at index i, then HCOL(j) is assigned the value i. When a Householder transformation is computed to eliminate, for example, $x(1)$ and $x(2)$ into $x(3)$, not only is the vector u computed as above, but also x is overwritten by Hx.

Similarly, each Givens rotation can be specified by a two-dimensional vector. Each such vector is stored in a column of the $2 \times g$ array QG, where g represents the maximum number of rotations expected. The associated index is stored in the corresponding element of the array GCOL. The Givens rotations are computed and applied by the **BLAS** routines **DROTG** and **DROT**, respectively.

The subprogram first computes $\tilde{F} = \tilde{P}^T F_1 \tilde{Q}$, and then F_1 by applying $\tilde{P}$ and $\tilde{Q}^T$ as described at the end of Section 2. At this point it is worth noting that, since the staircase form (4) is not computed by **DMEVAS**, transformations U and T in (12) are not applied in **DMEVAS**, but later on in **DBKTRN**.

The transformation $\tilde{P}^T$ consists entirely of rotations and is stored in QG beginning at a high column index and progressing toward lower column indices. Since the individual rotations of $\tilde{P}^T$ apply to only part of F_1, the associated index of F_1 is stored in the vector FCOL. $\tilde{Q}$ consists of both rotations and Householder transformations computed in each deflation step. The rotations and Householder transformations of $\tilde{Q}$ are both stored by increasing the column index beginning at column 1 in QG and QH, respectively. The end of each step is marked in the structures by setting negative in GCOL and HCOL the indices associated with the last rotation and reflector in that step.

If in a particular step a Householder transformation but no rotation is required, a dummy rotation is inserted into QG and recognized by its associated index in GCOL, which is given the value zero. Similarly, if a rotation but no Householder transformation is required, a dummy Householder transformation is introduced with associated index equal to zero placed in HCOL. These maneuvers facilitate the application of $\tilde{Q}^T$.

4.4 External Subroutines

The subroutines **DHHLDR**, **DHHRFL**, **DTINVB**, **D1NRMU**, **D1NRMA**, and **DABORT** are supporting routines for **DMEVAS**. In addition, the **LAPACK** routines **DTRTRS**, **DTRCON**, and **DLAMCH** and the **BLAS** routines **DASUM**, **DCOPY**, **DDOT**, **DROTG**, **DROT**, and **DSWAP** are also used. The demonstration program supplied also uses **BLAS** routine **DGEMM** and the **EISPACK** routines **BALANCE**, **ELMHES**, and **HQR**. These routines are supplied for convenience, but local libraries may be preferred when available.

4.5 Subroutines **DSTAIR** and **DBKTRN**

The heading of the subroutine **DSTAIR** and the declarations of its arguments are as follows:

```
      SUBROUTINE DSTAIR    (N, M, A, LDA, B, LDB, KMAX, KSTAIR,
     &                      ITRNSF, RTRNSF, IWORK, RWORK,
     &                      TOL, IWARN, IERR)
      INTEGER              N, M, KMAX, LDA, LDB, IWARN, IERR,
     &                     KSTAIR(*), ITRNSF(*), IWORK(*)
      DOUBLE PRECISION     A(LDA, *), B(LDB, *), RTRNSF(*),
     &                     RWORK(*), TOL
```

On input the leading $N \times N$ and $N \times M$ parts of the arrays A and B, respectively, contain the system matrices A and B in any form. On exit the same arrays contain the matrices A_1 and B_1 in staircase form (4). **ITRNSF** and **RTRNSF** have $M + 2N + 3 + \max(M, N)(M + 1)/2$ and $N(N + 1)/2 + \max(M, N)(M + 1)/2$ elements, respectively. They contain information required by **DBKTRN** pertaining to the transformations performed on A and B. On exit the warning and error arguments **IWARN** and **IERR** are either zero, meaning that nothing suspicious was detected, or they have values that may be interpreted as follows:

—**IWARN** = 1: The rank of B_1 or the rank of some block subdiagonal element of the staircase form of A_1 is less than **TOL**. The system is therefore uncontrollable to machine precision.

—**IERR** < 0: **IERR** $= -j$ indicates a problem with the jth argument on input. Specifically,

IERR $= -1$; on entry, $N < 1$;
IERR $= -2$; on entry, $M < 1$;
IERR $= -4$; on entry, LDA $< N$;
IERR $= -6$; on entry, LDB $< N$.

The heading of the subroutine **DBKTRN** and the declarations of its arguments are as follows:

SUBROUTINE DBKTRN (N, M, F, LDF, ITRNSF, RTRNSF, RWORK, IERR)
INTEGER N, M, LDF, ITRNSF(*), IERR
DOUBLE PRECISION F(LDF, *), RTRNSF(*), RWORK(*)

On input the leading $M \times N$ part of the array F contains the matrix F_1 as it is produced by the subroutine **DMEVAS**. On exit the same array contains the matrix $F = UF_1T^T$ in (12). **LDF** is the row dimension of array F, as declared in the calling program, **LDF** $\geq M$. **ITRNSF** and **RTRNSF** contain the transformation information as computed by **DSTAIR**. Unless the routine detects an error, **IERR** contains 0 on exit. If **IERR** $\neq 0$, the following list gives its meaning:

—**IERR** < 0: **IERR** $= -j$ indicates a problem with the jth argument on entry. Specifically,

IERR $= -1$; on entry, $N < 1$;
IERR $= -2$; on entry, $M < 1$;
IERR $= -4$; on entry, LDF $< M$.

A detailed example on how to use subroutines **DSTAIR**, **DMEVAS**, and **DBKTRN** to solve an EAP may be found in the demonstration program **DDEMO**, which is included in the accompanying software.

ACKNOWLEDGMENTS

We are grateful to the referees and the associate editor for a thorough review. We also thank Sharon Hynes for typesetting the original manuscript.

REFERENCES

DEMMEL, J. W. 1987. On condition numbers and the distance to the nearest ill-posed problem. *Numer. Math. 51*, 3 (July), 251–289.

GOLUB, G. H. AND VAN LOAN, C. F. 1989. *Matrix Computations*. 2nd ed. John Hopkins University Press, Baltimore, Md.

KAUTSKY, J., NICHOLS, N. K., AND VAN DOOREN, P. 1985. Robust pole assignment in linear state feedback. *Int. J. Control 41*, 5 (May), 1129–1155.

LAWSON, C., HANSON, R., KINCAID, D., AND KROGH, F 1979. Basic linear algebra subprograms for Fortran usage *ACM Trans Math. Softw. 5*, 3 (Sept.), 308–323.

LUENBERGER, D. G. 1979. *Introduction to Dynamic Systems.* Wiley, New York.

MIMINIS, G. S. AND PAIGE, C. C. 1982. An algorithm for pole assignment of time invariant multi-input linear systems. In *Proceedings of the 21st IEEE Conference on Decision and Control* (Orlando, Fla., Dec.). Vol. 1. IEEE, New York, 62–67.

MIMINIS, G. S. AND PAIGE, C. C. 1988. A direct algorithm for pole assignment of time-interval multiinput linear systems using state feedback. *Automatica 24*, 3 (May), 343–356.

MIMINIS, G. S. AND PAIGE, C. C. 1994. A QR-like approach for the eigenvalue assignment problem In *Proceedings of the 2nd Hellenic European Conference on Mathematics and Informatics* (Athens, Greece, Sept. 22–24).

MIMINIS, G. S. AND REID, M. 1993. A FORTRAN subroutine to solve the eigenvalue assignment problem for single-input systems. *ACM Trans Math. Softw 19*, 2 (June), 224–232.

PAIGE, C. C. 1981. Properties of numerical algorithms related to computing controllability. *IEEE Trans. Autom. Control AC-26*, 1 (Feb.), 130–138.

PATEL, R. V. AND MISRA, P. 1984. Numerical algorithms for eigenvalue assignment by state feedback. *Proc. IEEE 72*, 12 (Dec.), 1755–1764.

PETKOV, P., CHRISTOV, N , AND KONSTANTINOV, M. 1986. A computational algorithm for pole assignment of linear multi-input systems. *IEEE Trans. Autom Control AC-31*, 11 (Nov.), 1044–1047.

VAN DOOREN, P. 1981. The generalized eigenstructure problem in linear system theory *IEEE Trans. Autom. Control AC-26*, 1 (Feb.), 111–129

VARGA, A. 1981. A Schur method for pole assignment *IEEE Trans. Autom. Control AC-26*, 2 (Apr.), 517–519.

WONHAM, W. M. 1967. On pole assignment in multi-input controllable linear systems. *IEEE Trans. Autom Control AC-12*, 6 (Dec.), 660–665.

Received April 1991; revised February 1994, June 1994, and July 1994; accepted July 1994