A Fortran subroutine for column reduction of polynomial matrices

Eindhoven University of Technology
Department of Mathematics and Computing Science

A Fortran subroutine for column reduction of polynomial matrices

by

A.J. Geurts and C. Praagman

EUT Report 94-WSK-01
Eindhoven, June 1994

Authors' affiliations:

A.J. Geurts, Department of Mathematics and Computing Science,

Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands;

E-mail address: wstasli@urc.tue.nl

C. Praagman, Department of Econometrics,

University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands;

E-mail address: c.praagman@eco.rug.nl

Department of Mathematics and Computing Science

Eindhoven University of Technology

P.O. Box 513

5600 MB Eindhoven, The Netherlands

ISSN 0167-9708

Coden: TEUEDE

A Fortran subroutine for column reduction

of polynomial matrices

by

A.J. Geurts and C. Praagman

Abstract

In this report we describe a subroutine that takes an arbitrary polynomial matrix P as input, and yields on output a unimodular polynomial matrix U and a column reduced polynomial matrix R such that $PU = R$. The subroutine is based on the algorithm described in the paper by Neven and Praagman. The subroutine was run on four different computers, with comparable results. We found examples in which the routine behaves well, as well as examples in which the routine performs poorly, if no precautions are taken. We provide both kinds of examples and discuss the cause of the behavior of the routine. From these considerations a guideline for the use of the routine is derived.

Keywords. Polynomial matrix, column reduced, numerical method, Fortran subroutine.

AMS subject classifications. 65F30 – 15A23 – 15A22 – 15A24 – 15A33 – 93B10 – 93B17 – 93B25.

Acknowledgements. We would like to thank Dr. Alan Brown (NAG) and Herman Willemsen (TUE) for testing the routines on different computers.
We would like to thank Prof.dr. Gerhard Veltkamp for carefully reading the manuscript and making valuable suggestions.

Contents

1 Introduction

In systems theory polynomial matrices play a dominant role, for instance in the description of input-output systems:

$$Q_1(\tfrac{d}{dt})y = Q_2(\tfrac{d}{dt})u \ ,$$

where Q_1 and Q_2 are polynomial matrices of appropriate size, or in AR-descriptions:

$$Q(\tfrac{d}{dt})w = 0 \ .$$

In general the information these polynomial matrices contain can be redundant, in which case the polynomial matrices may be replaced by others of lower degree or smaller size without changing the behavior of the systems, i.e. without changing the set of trajectories satisfying the equations.

For reasons of simplicity, or for the possibility of finding state space descriptions, a minimal description can be convenient. Depending on the goal one has in mind, minimality can of course have a different meaning, but in important situations row or column reduced descriptions supply these minimal descriptions. This motivates the search for column reduced polynomial matrices equivalent to a non-column reduced original one. Examples of the importance of column or row reduced polynomial matrices may be found in the book of Kailath[6] or in the work of Willems [14,15,16] on behaviors. Note that P is row reduced if and only if its transpose P' is column reduced, so the problems of row reduction and column reduction are equivalent.

The algorithm underlying the subroutine has been developed in several stages: the part of the algorithm in which a minimal basis of the right null space of a polynomial matrix is calculated is an adaptation of the algorithm described in Beelen[1]. The original idea of the algorithm is described in Beelen, van den Hurk, and Praagman [2], and successive improvements have been reported in Neven[7], Praagman [10,11]. The paper by Neven and Praagman [8] gives the algorithm in the most general form, using iterations and exploiting the special structure of the problem in calculating the kernel of a polynomial matrix. The subroutine we describe here is an implementation of the algorithm in the latter paper.

The algorithm was supposed to perform well in all cases. But it turned out that the routine has difficulties in some special cases, in which the original polynomial matrix contains entries of different magnitude. We give examples in Section 7, and discuss the difficulties in detail in Section 8.

2 Preliminaries

Let us start with introducing some notations: Let $P \in \mathbf{R}^{m\times n}[s]$. Then $d(P)$, the *degree* of P, is defined as the maximum of the degrees of the entries of P, and $d_j(P)$, the *j-th column degree* of P, as the maximum of the degrees in the j-th column. $\delta(P)$ is the array of integers obtained by arranging the column degrees of P in non-decreasing order.

The *leading column coefficient matrix* of P, $\Gamma_c(P)$, is the constant matrix obtained by taking from column j the coefficients of the term with degree $d_j(P)$. Let $P_j = \sum_{k=0}^{d_j(P)} P_{jk}s^k$ be the j-th column of P, then the j-th column of $\Gamma_c(P)$ equals

$$\Gamma_c(P)_j := P_{jd_j(P)} \; .$$

Definition 1. *Let $P \in \mathbf{R}^{m\times n}[s]$. P is called* column reduced, *if there exists a permutation matrix T, such that $P = (\;0, \quad P_1\;)T$, where $\Gamma_c(P_1)$ has full column rank.*

Remark. Note that we do not require $\Gamma_c(P)$ to be of full column rank. In the literature there is some ambiguity about column properness and column reducedness. We follow here the definition of Willems [14]: P is *column proper* if $\Gamma_c(P)$ has full column rank, and column reduced if the conditions in the definition above are satisfied.

A square polynomial matrix $U \in \mathbf{R}^{n\times n}[s]$ is *unimodular* if $\det(U) \in \mathbf{R}\backslash\{0\}$ or, equivalently, if U^{-1} exists and is also polynomial.

It is well known that every regular polynomial matrix is unimodularly equivalent to a column proper matrix, see Wolovich [17]. Kailath [6] states that the above result can be extended to polynomial matrices of full column rank without changing the proof. In fact the proof in [18] is sufficient to establish that any polynomial matrix is equivalent to a column reduced matrix. Furthermore, Wolovich' proof implies immediately that the column degrees of the column reduced polynomial matrix do not exceed those of the original matrix, see Neven and Praagman [8].

Theorem 1. *Let $P \in \mathbf{R}^{m\times n}[s]$, then there exists a $U \in \mathbf{R}^{n\times n}[s]$, unimodular, such that $R := PU$ is column reduced. Furthermore $\delta(R) \leq \delta(P)$ totally.*

Although the proof of this theorem in the above sources is constructive, it is not suited for practical computations, for reasons explained in a paper by Van Dooren [13]. In Section 7 we give an example (example 3) that illustrates this point. In Neven and Praagman [8] an alternative, constructive proof is given on which the algorithm, underlying the subroutine we describe here, is based.

The most important ingredient of the algorithm is the calculation of a *minimal basis* of the right null space of a polynomial matrix associated to P.

Definition 2. *Let M be a submodule of $\mathbf{R}^n[s]$. Then $Q \in \mathbf{R}^{n\times r}[s]$ is called a* minimal basis *of M if Q is column proper and the columns of Q span M.*

Note that if $Q(s)$ has full column rank for all $s \in \mathbf{C}$, then M is a direct summand of $\mathbf{R}^n[s]$, so in that case Q is a minimal polynomial basis in the sense of Forney [4] or Beelen [1].

In the algorithm a minimal basis is calculated for the module

$$\ker(P,\ -I_m) := \{\{v \in \mathbf{R}^{n+m}[s] \mid (P,\ -I_m)v = 0\}\ ,$$

see [2]. Here and in the sequel I_m will denote the identity matrix of size m.

The first observation is that if $(U',\ R')'$ is such a basis, with $U \in \mathbf{R}^{n\times n}[s]$, then U is unimodular, see [2,8].

Of course, $R = PU$, but although $(U', \ R')'$ is minimal and hence column reduced, this does not necessarily hold for R. Take for example

$$P(s) = \begin{pmatrix} 1 & s \\ 0 & 1 \end{pmatrix} .$$

Then

$$\begin{pmatrix} U(s) \\ R(s) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & s \\ 0 & 1 \end{pmatrix}$$

is a minimal basis for $\ker(P, \ -I_m)$, but R is clearly not column reduced.

The next observation is that if $(U', \ R')'$ is a minimal basis for $\ker(P, \ -I_m)$, then $(U', \ s^b R')'$ is a basis for $\ker(s^b P, \ -I_m)$, but not necessarily a minimal basis. Especially, if $b > d(U)$, $(U', \ s^b R')'$ is minimal if and only if R' is column reduced. On the other hand, for any minimal basis $(U_b', \ R_b')'$ of $\ker(s^b P, \ -I_m)$, R_b' is divisible by s^b, and $PU_b = s^{-b} R_b'$. In [8] it is proved that for $b > (n-1)d(P)$, the calculation of a minimal basis of $\ker(s^b P, \ -I_m)$ yields a pair (U_b, R_b) in which R_b is column reduced.

3 Linearization

The calculation of $\ker(s^b P, \ -I_m)$ is done in the spirit of the procedure explained in [1]: we calculate a minimal basis of the kernel of the following linearization of $(s^b P, \ -I_m)$.

Let P be given by $P(s) = P_d s^d + P_{d-1} s^{d-1} + \ldots + P_0$. Define

$$
\begin{aligned}
H_b(s) &= sA_b - E_b \\
&= \begin{pmatrix}
sP_d & -I_m & 0 & \cdots & & & \cdots & 0 \\
sP_{d-1} & sI_m & -I_m & & & & & \vdots \\
\vdots & 0 & \ddots & \ddots & & & & \\
sP_0 & & & sI_m & -I_m & & & \\
0 & & & & \ddots & \ddots & & \vdots \\
\vdots & & & & & & & 0 \\
0 & & & & & & sI_m & -I_m
\end{pmatrix},
\end{aligned}
$$

where $A_b, E_b \in \mathbf{R}^{m_a \times n_a}$, with $m_a = (d+b)m$, and $n_a = n + m_a$.

With

$$
C_b(s) := \begin{pmatrix}
I_m & 0 & \cdots & & 0 \\
sI_m & I_m & & & \\
\vdots & \ddots & \ddots & & 0 \\
& & & & \\
s^{b+d-1} I_m & & & sI_m & I_m
\end{pmatrix}
$$

we see that

$$
C_b(s)H_b(s) = \begin{pmatrix}
sP_d & -I_m & 0 & \cdots & 0 \\
s^2 P_d + sP_{d-1} & 0 & -I_m & & \vdots \\
\vdots & & & \ddots & 0 \\
s^b P & & & 0 & -I_m
\end{pmatrix},
$$

so if V is a basis of $\ker(H_b)$, then

$$
\begin{pmatrix} U \\ R \end{pmatrix} := \begin{pmatrix} I_n & 0 & \cdots & 0 \\ 0 & \cdots & 0 & I_m \end{pmatrix} \cdot V
$$

is a basis for $\ker(s^b P, \ -I_m)$ [1]. In [8] it is proved that $\delta(V) = \delta((U', \ R')')$ and that V is minimal if and only if $(U', \ R')'$ is minimal. So the problem is to calculate a minimal basis for $\ker(H_b)$.

4 The calculation of a minimal basis

A minimal basis is calculated by transforming the pencil H_b by orthogonal pre- and post–transformations to a form where E_b has not changed and A_b is in *upper staircase* form.

Definition 3. *A matrix $A \in \mathbf{R}^{m_a \times n_a}$ is in* upper staircase *form if there exists an increasing sequence of integers s_i, $1 \leq s_1 < s_2 < \ldots < s_r \leq n_a$, such that $A_{i,s_i} \neq 0$ and $A_{ij} = 0$ if $i > r$ or $j < s_i$. The elements A_{i,s_i} are called the* pivots *of the staircase form.*

Note that $i \leq s_i$ and that the submatrix of the first i rows and the first s_i (or more) columns of A is right invertible for $1 \leq i \leq r$.

In [1,8] it has been shown that there exist orthogonal, symmetric matrices $Q_1, Q_2, \ldots, Q_r$ such that

$$\bar{A}_b := Q_r \ldots Q_1 A_b Q_1^{<n>} \ldots Q_r^{<n>}$$

is in upper staircase form. The matrices Q_k are elementary Householder reflections, and we define $Q_k^{<n>} := \mathrm{diag}(I_n, Q_k)$. Note that E_b is not changed by this transformation:

$$E_b := Q_r \ldots Q_1 E_b Q_1^{<n>} \ldots Q_r^{<n>} \ .$$

We partition $\bar{A}_b$ as follows:

$$\bar{A}_b = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \cdots & & A_{1,l+2} \\ 0 & A_{22} & A_{23} & & & \vdots \\ \vdots & 0 & \ddots & \ddots & & \\ 0 & & & A_{ll} & A_{l,l+1} & A_{l,l+2} \\ 0 & & & & 0 & A_{l+1,l+2} \end{pmatrix},$$

with $A_{jj} \in \mathbf{R}^{m_j \times m_{j-1}}$ right invertible and in upper staircase form, $j = 1, \ldots, l$, and $A_{j,j+1}$ a square matrix, for $j = 1, \ldots, l+1$. We take $m_0 = n$. Then the dimensions $m_j, j = 1, \ldots, l$, are uniquely determined and

$$
\begin{aligned}
\bar{H}_b &:= s\bar{A}_b - E_b \\
&= \begin{pmatrix}
sA_{11} & sA_{12}-I & sA_{13} & \cdots & & sA_{1,l+2} \\
0 & sA_{22} & sA_{23}-I & & & \vdots \\
\vdots & 0 & \ddots & \ddots & & \\
0 & & & sA_{ll} & sA_{l,l+1}-I & sA_{l,l+2} \\
0 & & & & 0 & sA_{l+1,l+2}-I
\end{pmatrix}.
\end{aligned}
$$

Let A_b^* be the submatrix of $A_b^{(k-1)} := Q_{k-1}\ldots Q_1 A_b Q_1^{<n>}\ldots Q_{k-1}^{<n>}$ obtained by deleting the first $k-1$ rows, and let s_k be the column index of the first nonzero column in A_b^*. Then Q_k transforms this (sub)column into the first unit vector and lets the first $k-1$ rows and the first $s_k - 1$ columns of $A_b^{(k-1)}$ invariant. Consequently, postmultiplication with $Q_k^{<n>}$ leaves the first $n+k-1$ columns of $A_b^{(k-1)}$ unaltered.

Because of the staircase form of $\bar{H}_b$ it is easy to see that the equation $\bar{H}_b y = 0$ has $m_{i-1} - m_i$ independent solutions of the form

$$
y := \begin{pmatrix}
y_{11} & y_{12} & \cdots & y_{1i} \\
0 & y_{22} & & \\
 & & \ddots & \\
 & & & y_{ii} \\
0 & & & 0
\end{pmatrix}
\begin{pmatrix}
1 \\ s \\ \vdots \\ s^{i-1}
\end{pmatrix},
$$

where $y_{ii} \in \mathbf{R}^{m_{i-1}}$ is a null vector of A_{ii}, and $y_{jk} \in \mathbf{R}^{m_{j-1}}$, $k = j, \ldots, i$. Clearly $v := Q_1^{<n>}\cdots Q_r^{<n>} y$ is then a null vector of H_b and taking the top and bottom part of v yields a column of U_b and R_b, respectively, of degree $i-1$. Note that this implies that $l \leq b+d+1$, since $(I_n,\ s^b P')'$ is a basis of $\ker(s^b P,\ -I_m)$ and the degrees of the minimal bases of $\ker(H_b)$ and $\ker(s^b P,\ -I_m)$ are the same, see the end of Section 3.

5 Increasing b

The computational effort to calculate a minimal basis for $\ker(s^b P,\ -I_m)$ increases quickly with the growth of b. From experiments, however, we may assume that in many cases a small b already leads to success. Therefore the algorithm starts with $b = 1$ and increases b by one until a column reduced R_b has been found. With the transition from b to $b+1$ the computations need not start from scratch, as we will explain in this section.

The transformation of A_b into $\bar{A}_b$ is split up into steps, where in the j-th step the diagonal block matrix A_{jj} is formed, $j = 1, \ldots, l$. Let μ_j denote the row index of the first row of A_{jj}. Then A_{jj} is formed by the Householder reflections $Q_{\mu_j}, \ldots, Q_{\mu_{j+1}-1}$. Let $N_k \in \mathbf{R}^{m\times(n+(d+k)m)}$ denote the matrix (0 ,..., 0, I_m). Then

$$A_{b+1} = \begin{pmatrix} A_b & 0 \\ N_b & 0 \end{pmatrix},$$

as well as

$$A_{b+1} = \begin{pmatrix} A_1 & 0 & 0 \\ N_1 & 0 & 0 \\ 0 & I_{(b-1)m} & 0 \end{pmatrix}.$$

Observe that the first n columns of A_{b+1} are just the first n columns of A_1 augmented with zeros. Since postmultiplication with any $Q^{<n>}$ does not affect the first n columns, it is clear that the reflection vectors of the first $\mu_2 - 1$ Householder reflections of A_{b+1} (the ones involved in the computation of A_{11}) are the reflection vectors of the first $\mu_2 - 1$ Householder reflections of A_1 augmented with zeros. Let $K_{1;1} := Q_{\mu_2-1}\cdots Q_1$ be the orthogonal transformation of the first step of the transformation for A_1. Then $K_{1;b+1} := \operatorname{diag}(K_{1;1}, I_{bm})$ is the corresponding transformation for A_{b+1} and the first step can be described by

$$\begin{aligned} K_{1;b+1}A_{b+1}K_{1;b+1}^{<n>} &= \begin{pmatrix} K_{1;1}A_1K_{1;1}^{<n>} & 0 & 0 \\ N_1K_{1;1}^{<n>} & 0 & 0 \\ 0 & I_{(b-1)m} & 0 \end{pmatrix} \\ &= \begin{pmatrix} K_{1;2}A_2K_{1;2}^{<n>} & 0 & 0 \\ N_2 & 0 & 0 \\ 0 & I_{(b-2)m} & 0 \end{pmatrix}, \end{aligned}$$

where $K_{1;j}^{<n>}$ has the obvious meaning. From this we see that after the first step the second block column of width $\mu_2 - 1$, i.e. the block column from which A_{22} will be acquired, is exactly the corresponding block column of A_2 after the first step augmented with zeros, if $b \geq 2$. In the second step, postmultiplication with the Householder reflections $Q_k^{<n>}$, for $k = \mu_2, \ldots, \mu_3 - 1$, does not affect the first $n + \mu_2 - 1$ columns. Therefore, the argumentation for the first step applies, mutatis mutandis, for the second step, if $b \geq 3$. As a consequence, it can be concluded by induction that the transformations for the j-th step for A_{b+1} and A_b are related by

$$K_{j;b+1} = \operatorname{diag}(K_{j;b},\ I_m), \qquad j = 1, \ldots, b,$$

and that we can start the algorithm for A_{b+1} with A_b transformed by $K_{j,b}$, for $j = 1, \ldots, b$, augmented with $N_b K_{b,b}^{<n>}$ at the bottom and m zero columns on the right.

6 Description of the algorithm

The algorithm consists of:

- An outer loop in b, running from 1 to $(n-1)d + 1$ at most. The termination criterion is that the calculated R_b is column reduced.
- An intermediate loop in i, running from b to $b + d + 1$ at most, in which A_{ii} is determined and the null vectors of H_b of degree $i - 1$ are calculated. The precondition is that A_{jj} and the null vectors of H_b of degree $j - 1$, for $j = 1, \ldots, i - 1$, have been computed and are available. The termination criterion is that the submatrix of computed columns of R_b is not column reduced, or that all the null vectors of H_b have been found.
- An inner loop in k, running from $n + \mu_{i-1}$ to $n + \mu_i - 1$, ($\mu_0 := -n$) indexing the columns of A_{ii}. For each k, either a Householder reflection is generated and applied or a null vector of degree $i - 1$ is computed. If a null vector has been found, then U_b and R_b are extended with one column and R_b is checked on column reducedness. The loop is terminated after $k = n + \mu_i - 1$ or if the extended R_b is not column reduced.

The algorithm uses a tolerance below which the matrix elements are considered to be zero. Thus the tolerance is used to determine the rank of the diagonal blocks A_{ii} from which the null vectors of H_b and consequently the computed solution follows. The choice of the tolerance does not influence the *accuracy* of the computed solution.

The algorithm has been implemented in the subroutine **COLRED**. The subroutine is written according to the standards of SLICOT, see [9], with a number of auxiliary routines. It is based on the BLAS [3,5], and on similar routines from the NAG library [12, Chapter F06].

For programming details the reader is referred to the Appendix which contains the full text of the routines as well as an example program. The example program uses also two general routines: **MULTPM**, for the addition and multiplication of polynomial matrices, and **PRMAPO**, for printing a polynomial matrix.

7 Examples

In this section we describe a few examples. All examples were run on four computers, a VAX-VMS, a VAX-UNIX, a SUN and on a 386 personal computer. The numerical values that we present here were produced on the VAX-VMS computer. Its machine precision is $2^{-56} \approx 1.4 * 10^{-17}$.

The first example is taken from the book of Kailath [6], and has been discussed before in [2] and [8].

Example 1. The polynomial matrix P is given by

$$P(s) = \begin{pmatrix} s^4 + 6s^3 + 13s^2 + 12s + 4 & -s^3 - 4s^2 - 5s - 2 \\ 0 & s + 2 \end{pmatrix} .$$

In Kailath [6, p.386] we can find (if we correct a small typo) that $PU_0 = R_0$, with

$$U_0(s) = \begin{pmatrix} 1 & 0 \\ s + 2 & 1 \end{pmatrix} ,$$

$$R_0(s) = \begin{pmatrix} 0 & -(s^3 + 4s^2 + 5s + 2) \\ s^2 + 4s + 4 & s + 2 \end{pmatrix} .$$

Clearly R_0 is column reduced, and U_0 unimodular. This example was also treated in [2]. The program, with a prescribed tolerance of 10^{-12}, yields the following solution

$$U(s) = \begin{pmatrix} \alpha & -\beta s - \gamma \\ \alpha(s+2) & -\beta s^2 - \delta s \end{pmatrix},$$

$$R(s) = \begin{pmatrix} 0 & -2\gamma(s^3+4s^2+5s+2) \\ \alpha(s^2+4s+4) & (-\beta s^2 - \delta s)(s+2) \end{pmatrix},$$

with $\alpha = 7.302027$, $\beta = 37.43234s$, $\gamma = 31.87083$ and $\delta = 2\beta + \gamma$.

It is easily checked that $PU - R = O(10^{-13})$, and that U is unimodular:

$$U(s) = \begin{pmatrix} 1 & 0 \\ s+2 & 1 \end{pmatrix} \begin{pmatrix} \alpha & -\beta s - \gamma \\ 0 & 2\gamma \end{pmatrix}.$$

This solution is found without iterations, so for $b = 1$, and equals the solution found in [2].

As already mentioned in [2] one of the main motivations for the iterative procedure, that is starting with a small b and increasing b until the solution is found, is the (experimental) observation that in most examples increasing b is unnecessary. The following example, also treated in [2,8] is constructed especially to show that sometimes a larger b is required.

Example 2.

$$P(s) = \begin{pmatrix} s^4 & s^2 & s^6+1 \\ s^2 & 1 & s^4 \\ 1 & 0 & 1 \end{pmatrix}.$$

Note that this matrix is unimodular and hence unimodularly equivalent to a constant, invertible matrix. The program, run with the tolerance set to 0.6×10^{-14}, yields no column reduced R for $b \leq 4$. For $b = 5$ the resulting U and R are

$$U(s) = \begin{pmatrix} 1 & -1 & \alpha s^2 \\ -s^2 & -s^4+s^2 & \alpha(-s^6+s^4-1) \\ 0 & 1 & \alpha s^2 \end{pmatrix}$$

$$R(s) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & -\alpha \\ 1 & 0 & 0 \end{pmatrix},$$

where $\alpha = 1.702939\ldots$. The residual matrix satisfies: $PU - R = O(10^{-15})$.

The above examples behave very well, in fact they are so 'regular' that also the algorithm based on Wolovich's proof of Theorem 1 (from now on called the Wolovich algorithm) yields reliable answers. In a forthcoming report we will compare the results of both algorithms to a greater extent. Here we restrict ourselves to giving two more examples, for which the Wolovich algorithm yields nonsensical answers.

Example 3. In the third example we take for P:

$$P(s) = \begin{pmatrix} s^3 + s^2 & \varepsilon s + 1 & 1 \\ 2s^2 & -1 & -1 \\ 3s^2 & 1 & 1 \end{pmatrix},$$

with ε a small parameter. Calculation by hand immediately shows that taking U equal to

$$U(s) = \begin{pmatrix} 1 & 0 & 0 \\ -\delta s^2 & \delta & 0 \\ \delta s^2 & -\delta & 1 \end{pmatrix},$$

with $\delta = \varepsilon^{-1}$, yields an R (equal to PU) given by

$$R(s) = \begin{pmatrix} s^2 & s & 1 \\ 2s^2 & 0 & -1 \\ 3s^2 & 0 & 1 \end{pmatrix}.$$

Clearly, R is column reduced, and U unimodular. Note that U contains large elements as ε is small, a feature that will also be seen in the answers provided by the program.

For small values of ε the Wolovich algorithm behaves badly (see our forthcoming report). If we take $\varepsilon = 10^{-2}$ and set the tolerance to 10^{-14}, our algorithm yields

$$U(s) = \begin{pmatrix} 0 & 0 & 2\beta \\ 0 & -3\alpha\delta & -\beta\delta(2s^2+s) \\ \delta & \alpha s & \beta((2\delta-1)s^2+2\delta s) \end{pmatrix},$$

$$R(s) = \begin{pmatrix} \delta & -\alpha(2s+3\delta) & \beta\delta s \\ -\delta & -\alpha(s+3\delta) & \beta(5s^2-\delta s) \\ \delta & \alpha(s-3\delta) & \beta(5s^2+\delta s) \end{pmatrix},$$

with $\alpha = 0.4082483$ and $\beta = 0.14144214$.

For smaller ε the tolerance has to be increased to obtain an answer with the same structure as above, e.g. for $\varepsilon = 10^{-4}$ and $\varepsilon = 10^{-6}$ the tolerance must at least be 10^{-12} and 10^{-10}, respectively. In all cases $PU - R = O(10^{-16}\|U\|)$. In the next section we will analyze this example in more detail.

Of course the first thought is that the occurrence of the small parameter ε is the cause of the problem, but the next example shows that not in all cases the occurrence of a small parameter leads to phenomena as in the previous example.

Example 4. In the fourth example we take for P:

$$P(s) = \begin{pmatrix} s^3+s^2+2s+1 & \varepsilon s^2+2s+3 & s^2+s+1 & s-1 \\ s-1 & -s+2 & 2s^2+s-1 & 2s+1 \\ s+3 & 2s-1 & -s^2-2s+1 & -s-2 \\ 1 & -1 & 3s+1 & 3 \end{pmatrix}.$$

For all ε with $0 < \varepsilon \leq 10^{-8}$, and the tolerance set to 10^{-16}, the program yields good results with $PU - R = O(10^{-16})$. For instance, with $\varepsilon = 10^{-8}$ the results are

$$U(s) = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 2.484367 \\ -0.665755 & 1.596887 & -0.204285 & 0.0 \\ 0.0 & 2.021052 & 0.133914 & 0.0 \\ 0.0 & 0.0 & -1.320416 & 0.0 \end{pmatrix} +$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.4225558 \\ 0.3328775\varepsilon & -0.7984436\varepsilon & 0.1021425\varepsilon & -0.0069700 \\ 0.0 & 2.021052 & -0.1339140 & 1.389642 \end{pmatrix} s +$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -1.242183 \\ -0.3328775\varepsilon & 0.7984436\varepsilon & -0.1021425\varepsilon & 0.0069700 \end{pmatrix} s^2 +$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.242183 \end{pmatrix} s^3 ,$$

and

$$R(s) = \begin{pmatrix} -1.997265 & 2.769609 & 0.8414748 & 2.484367 \\ -1.33151 & 5.214827 & -1.862900 & -2.484367 \\ 0.665755 & 3.617939 & 2.979030 & 7.453100 \\ 0.665755 & 3.617939 & -3.623048 & 2.484367 \end{pmatrix} +$$

$$\begin{pmatrix} -1.33151 & -0.848333 & -1.461158 & 2.304454 \\ 0.665755 & -1.596887 & -2.436546 & 3.035867 \\ -1.33151 & 3.193774 & 0.9118457 & 0.1206680 \\ 0.3328775\varepsilon & -0.7984436\varepsilon & 0.1021425\varepsilon & 4.584513 \end{pmatrix} s +$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 1.772774 \\ 0.0 & 0.0 & 0.0 & 4.444024 \\ 0.0 & 0.0 & 0.0 & 3.476937 \\ 0.0 & 0.0 & 0.0 & 1.242183 \end{pmatrix} s^2 .$$

The residual matrix $PU - R$ has entries smaller than 10^{-15}. In the next section we also revisit this example.

We also investigated the sensitivity of the algorithm to perturbations in the data. The conclusion, based on a number of experiments, is that if the tolerance is chosen such that the perturbations lie within the tolerance, then the program retrieves the results of the unperturbed system. This is well in agreement with the general feeling. We give one example.

Example 5.

$$P(s) = \begin{pmatrix} s^3+s^2+s & s^2+1 & 1 \\ s^3+2s^2+3s & s^2 & 1 \\ s^3+3s^2+s+1 & s^2+1 & 1 \end{pmatrix}.$$

The resulting R has column degrees $(0,0,2)$. Disturbing P with quantities in the order of 10^{-8} leads to the result that the disturbed P is column reduced if the tolerance is less than 10^{-8}. Setting the tolerance to 10^{-7} gives again an outcome in accordance with the unperturbed case:

$$U(s) = \begin{pmatrix} 0.0 & 0.0 & 0.7335386 \\ -0.6324555 & -0.5071715 & 0.0 \\ 0.0 & 0.7513652 & 0.0 \end{pmatrix} +$$
$$\begin{pmatrix} 0.0 & 0.0 & 0.3221301 \\ 0.0 & 0.0 & 0.1527825 \\ 0.0 & 0.0 & -1.285900 \end{pmatrix} s +$$
$$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.1256059 \\ 0.6324555 & 0.5071715 & -1.517319 \end{pmatrix} s^2 +$$
$$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.8863210 \end{pmatrix} s^3 +$$
$$\begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.1256059 \end{pmatrix} s^4$$

$$R(s) = \begin{pmatrix} -0.6324555 & 0.2441937 & 0.0 \\ 0.0 & 0.7513652 & 0.0 \\ -0.6324555 & 0.2441937 & 0.7335386 \end{pmatrix} +$$
$$\begin{pmatrix} 0.0 & 0.0 & -0.3995791 \\ 0.0 & 0.0 & 0.9147156 \\ 0.0 & 0.0 & -0.3995791 \end{pmatrix} s +$$
$$\begin{pmatrix} 0.0 & 0.0 & -0.6581750 \\ 0.0 & 0.0 & -0.0502423 \\ 0.0 & 0.0 & 0.8089021 \end{pmatrix} s^2 .$$

8 Discussion

First we examine example 3 of Section 7, namely

$$P_\varepsilon(s) = \begin{pmatrix} s^3 + s^2 & \varepsilon s + 1 & 1 \\ 2s^2 & -1 & -1 \\ 3s^2 & 1 & 1 \end{pmatrix},$$

The U_ε and R_ε that will result from the algorithm, if calculations are performed exactly, are

$$U_\varepsilon(s) = \begin{pmatrix} 0 & 0 & 2\beta \\ 0 & -3\alpha\delta & -\beta\delta(2s^2 + s) \\ \delta & \alpha s & \beta((2\delta - 1)s^2 + 2\delta s) \end{pmatrix},$$

$$R_\varepsilon(s) = \begin{pmatrix} \delta & -\alpha(2s + 3\delta) & \beta\delta s \\ -\delta & -\alpha(s + 3\delta) & \beta(5s^2 - \delta s) \\ \delta & \alpha(s - 3\delta) & \beta(5s^2 + \delta s) \end{pmatrix},$$

with $\alpha = \frac{1}{6}\sqrt{6}$, $\beta = \frac{1}{10}\sqrt{2}$ and $\delta = \varepsilon^{-1}$.

In Section 7 we saw that the tolerance for which this result is obtained by the routine is proportional to ε^{-1}. Close examination of the computations reveals that the computed $\hat{A}_b$ (see Section 4) gets small pivots, which cause growing numbers in the computation of the right null vectors until overflow occurs, and a breakdown of the process if the tolerance is too small. Scaling of a null vector, which at first sight suggests itself, may suppress the overflow and thereby hide the problem at hand. In this example the effect of scaling is that if ε tends to zero, then U_ε tends to a singular matrix and R_ε to a constant matrix of rank 1.

Is there any reason to believe that there exists an algorithm which yields an R continuously depending on ε in a neighborhood of 0? Observe that

- $\det(P_\varepsilon)(s) = -5\varepsilon s^3$, so P_ε is singular for $\varepsilon = 0$;
- the column degrees of R_ε are $(0,1,2)$ if $\varepsilon \neq 0$ and $(-1,0,3)$ if $\varepsilon = 0$ (We use the convention that the zero polynomial has degree -1).

We conclude that the entries of U_ε and R_ε do not depend continuously on ε in a neighborhood of $\varepsilon = 0$. Even stronger: *There do not exist families $\{V_\varepsilon\}$, $\{S_\varepsilon\}$, continuous in $\varepsilon = 0$ such that for all ε, V_ε is unimodular, S_ε column reduced, and $P_\varepsilon V_\varepsilon = S_\varepsilon$.*

Example 4, though at first sight similar, is quite different from example 3. Due to the fact that the third column of P minus s times its fourth column equals $(2s+1,-1,1,1)^t$, the term εs^2 in the element P_{12} is not needed to reduce the first column. As a consequence, the elements of U_ε and R_ε depend continuously on ε and no large entries occur. This feature is not recognized by the Wolovich algorithm. Perturbation of P_ε, for instance changing the (4,4) entry from 3 to $3+\delta$, destroys this property. The resulting matrix behaves similarly to example 3. To compare examples 3 and 4 we observe that in example 4

- $\det(P_{\varepsilon,\delta}) \neq 0$ for all values of ε and δ, so this property is not characteristic;
- in the unperturbed case, $\delta = 0$, the column degrees of R_ε are $(1,1,1,2)$ for all ε. If $\delta \neq 0$, then the column degrees of R_ε are $(1,1,2,2)$ for $\varepsilon \neq 0$ and $(1,1,1,3)$ if $\varepsilon = 0$. So here again we can conclude that in this case no algorithm can yield $U_{\varepsilon,\delta}$, $R_{\varepsilon,\delta}$ continuous in $\varepsilon = 0$. This is what we call a *singular* case.

Remark. Singularity in the sense just mentioned may appear in a more hidden form. For instance, if in example 4 the third column is added to the second, resulting in $P_{12} = (1+\varepsilon)s^2 + 3s + 4$, we get a similar behavior depending on the values of δ and ε. Though

ε in P_{12} is likely to be a perturbation, its effect is quite different from the effects of the perturbations in example 5.

For perturbations as in example 5 the tolerance should at least be of the order of magnitude of the uncertainties in the data to find out whether there is a non-column reduced polynomial matrix in the range of uncertainty. In cases like example 5 it may be wise to run the algorithm for several values of the tolerance.

9 Conclusions

In this report we described a subroutine which is an implementation of the algorithm developed by Neven and Praagman [8]. The routine asks for the polynomial matrix P to be reduced, and a tolerance. The tolerance is used for rank determination within the accuracy of the computations. Thus the tolerance influences whether or not the correct solution is found, but does not influence the accuracy of the solution.

We gave five examples. In all cases the subroutine performs satisfactorily, i.e. the computed solution has a residual matrix $PU - R = O(\|U\| * EPS)$, where EPS is the machine precision. Normally the tolerance should be chosen in accordance with the accuracy of the elements of P, with a lower bound (default value) of the order of EPS times the Frobenius norm of P. In some cases the tolerance has to be set to a larger value than the default value in order to get significant results. Therefore, in case of failure, or if there is doubt about the correctness of the solution, the user is recommended to run the program with several values of the tolerance.

At this moment we are optimistic about the performance of the routine. The only cases for which we had some difficulties to get the solution were what we called singular cases. As we argued in the last section, the nature of this singularity will frustrate in fact all algorithms. We believe, although we cannot prove it at this moment that the algorithm is numerically stable in the sense that the computed solution satisfies $\|PU - R\| = O(\|P\|\|U\|EPS)$.

References

[1] Th.G.J. Beelen, *New algorithms for computing the Kronecker structure of a pencil with applications to systems and control theory*, PhD thesis, Eindhoven University of Technology, Eindhoven, 1987.

[2] Th.G.J. Beelen, G.J.H.H. van den Hurk, and C. Praagman, A new method for computing a column reduced polynomial matrix, *Systems Control Lett.* **10**, 217-224 (1988).

[3] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh, Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software* **5**, 308-323 (1979).

[4] G.D. Forney, Minimal bases of rational vector spaces with applications to multivariable linear systems. *SIAM J. Control Optim.* **13**, 493-520 (1975).

[5] J.J. Dongarra, J.J. Du Croz, S.J. Hammerling, and R.J. Hanson, An extended set of Fortran basic linear algebra subprograms. *ACM Trans. Math. Software*, **14**, 1-17 (1988).

[6] T. Kailath, *Linear systems*, Prentice–Hall, Englewood Cliffs, 1980.

[7] W.H.L. Neven, Polynomial methods in systems theory, Master's thesis, Eindhoven University of Technology, Eindhoven, 1988.

[8] W.H.L. Neven and C. Praagman, Column reduction of polynomial matrices, *Linear Algebra Appl.* **188-189**, 569-589 (1993).

[9] Working Group on Software (WGS), *SLICOT: Implementation and Documentation Standards*, Volume 90-01 of *WGS- report*, WGS, Eindhoven/Oxford, May 1990.

[10] C. Praagman, Inputs, outputs and states in the representation of time series, in A. Bensoussan and J.L. Lions (Editors), *Analysis and optimization of systems*, pp. 1069-1078, Lecture Notes in Control and Information Sciences, 111, Springer, Berlin, 1988.

[11] C. Praagman, Invariants of polynomial matrices, in I. Landau (Editor), Proceedings of the first ECC Grenoble 1991, pp. 1274-1277, INRIA, 1991.

[12] The Numerical Algorithms Group Ltd., NAG Fortran Library, Mark 14, Oxford, 1990.

[13] P. van Dooren, The computation of Kronecker's canonical form of a singular pencil, *Linear Algebra Appl.* **27**, 103-140 (1979).

[14] J.C. Willems, From time series to linear systems: Part 1,2,3. *Automatica J. IFAC* **22/23**, 561-580, 675-694, 87-115 (1986/87).

[15] J.C. Willems, Models for dynamics, *Dynamics reported*, **2**, 171-269 (1988).

[16] J.C. Willems, Paradigms and puzzles in the theory of dynamical systems, *IEEE Trans. Automat. Control.* **36**, 259- 294 (1991).

[17] W.A. Wolovich, *Linear multivariable systems*, Springer Verlag, Berlin, 1978.

Appendix

This appendix contains the Fortran code of the example program that has been used to produce the examples in Section 7, and the subroutines. The interdependence of the subroutines is sketched in the subroutine tree in figure 1:

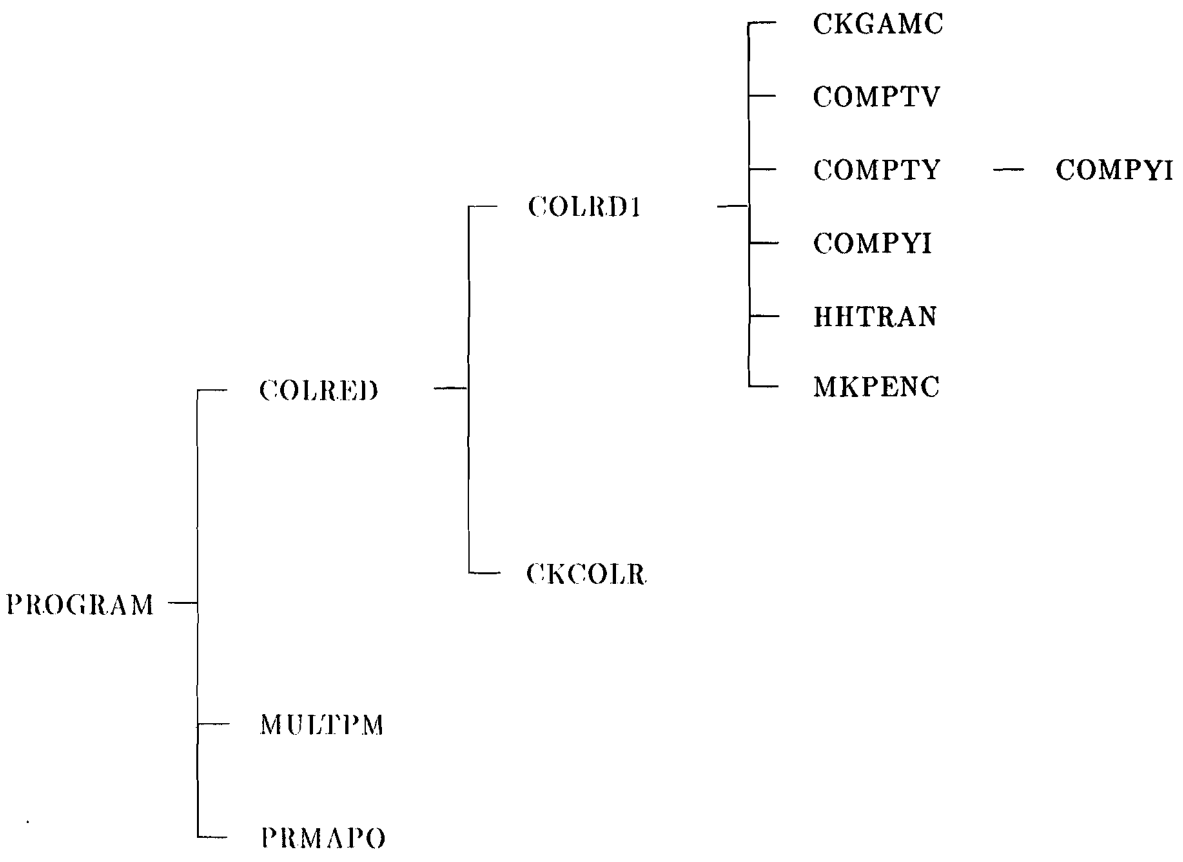

Figure 1: Subroutine tree for the example program

The example program

```
C
C     COLRED EXAMPLE PROGRAM TEXT
C
C     Version dd February 15, 1994.
C
      IMPLICIT NONE
C     .. Parameters ..
      INTEGER NIN, NOUT
      PARAMETER (NIN = 5, NOUT = 6)
      DOUBLE PRECISION ONE
      PARAMETER (ONE = 1.0D0)
C
      INTEGER DPMAX, MPMAX, NPMAX
      PARAMETER (DPMAX = 6, MPMAX = 10, NPMAX = 10)
      INTEGER ND1, LIWORK, LRWORK, DRMAX
      PARAMETER (ND1 = NPMAX * DPMAX + 1,
     *           LIWORK = 2 * ND1 * MPMAX + NPMAX + 1,
     *           LRWORK = (2*MPMAX+1) * MPMAX * ND1**2 +
     *                    (2*MPMAX*NPMAX + 4*MPMAX +NPMAX) * ND1 +
     *                    (2*MPMAX+3) * NPMAX,
     *            DRMAX = (NPMAX+1)*DPMAX + 1)
      INTEGER LDP1, LDP2, LDR1, LDR2, LDU1, LDU2
      PARAMETER (LDP1 = MPMAX, LDP2 = NPMAX, LDR1 = MPMAX,
     *           LDR2 = NPMAX, LDU1 = NPMAX, LDU2 = NPMAX)
C     .. Local Scalars ..
      INTEGER MP, NP, DP, DR, DU, I, J, K, IERR
      DOUBLE PRECISION TOL
C     .. Local Arrays ..
      INTEGER IWORK(LIWORK)
      DOUBLE PRECISION P(LDP1,LDP2,DPMAX+1), R(LDR1,LDR2,DRMAX),
     *                 U(LDU1,LDU2,ND1), RWORK(LRWORK)
      LOGICAL ZERCOL(NPMAX)
C     .. External Subroutines/Functions ..
      EXTERNAL COLRED, MULTPM, PRMAPO
C
C     .. Executable Statements ..
C
C     Read MP, NP, DP, TOL and next P(k), k = 0,....,DP row after row.
C
      WRITE(NOUT, FMT = 99999)
      READ(NIN, FMT = '()')
      READ(NIN, FMT = *) MP, NP, DP, TOL
C
      DO 20 K = 1, DP + 1
         READ(NIN, FMT = '()')
```

```
         DO 10 I = 1, MP
            READ(NIN, FMT = *) (P(I,J,K), J = 1, NP)
 10      CONTINUE
 20   CONTINUE
C
      WRITE(NOUT, FMT = 99998) DP, MP, NP, TOL
      CALL PRMAPO(MP, NP, DP, 5, NOUT, P, LDP1, LDP2, 'P', IERR)
C
      CALL COLRED(MP, NP, DP, P, LDP1, LDP2, DR, DU, R, LDR1, LDR2,
     *            U, LDU1, LDU2, ZERCOL, IWORK, RWORK, TOL, IERR)
C
      IF (IERR .EQ. 0) THEN
         WRITE (NOUT, FMT = 99997)
         CALL PRMAPO(NP, NP, DU, 5, NOUT, U, LDU1, LDU2, 'U', IERR)
C
         WRITE (NOUT, FMT = 99996)
         CALL PRMAPO(MP, NP, DR, 5, NOUT, R, LDR1, LDR2, 'R', IERR)
         WRITE (NOUT, FMT = 99995) (ZERCOL(J), J = 1, NP)
C
         CALL MULTPM(-ONE, MP, NP, NP, DP, DU, DR, P, LDP1, LDP2,
     *               U, LDU1, LDU2, R, LDR1, LDR2, RWORK, IERR)
         IF (DR .GE. 0) THEN
            WRITE (NOUT, FMT = 99994)
            CALL PRMAPO(MP, NP, DR, 5, NOUT, R, LDR1, LDR2, '(PU-R)',
     *                  IERR)
         ELSE
            WRITE (NOUT, FMT = 99993)
         END IF
      ELSE
         WRITE (NOUT, FMT = 99992) IERR
      END IF
      STOP
C
99999 FORMAT (' COLRED EXAMPLE PROGRAM RESULTS', /1X)
99998 FORMAT (' The input polynomial matrix:', //,
     *        ' P(s) = P(0) + P(1) * s + . . . + P(dp-1) *s**(dp-1)',
     *        '  + P(dp) * s**dp', //, ' with degree DP =', I2,
     *        ' and size MP =', I2, ', NP =', I2, '.', //,
     *        ' The tolerance is:', D10.3, /1X)
99997 FORMAT (' The unimodular polynomial matrix U(s):')
99996 FORMAT (' The column reduced polynomial matrix R(s):')
99995 FORMAT (' ZERCOL(j), j = 1, NP:', 10(L2))
99994 FORMAT (' The residual matrix P(s) * U(s) - R(s):')
99993 FORMAT (' PU - R is the ZERO polynomial matrix.')
99992 FORMAT (' COLRED has failed: IERR =', I2)
      END
```

The routine COLRED

```
      SUBROUTINE COLRED(MP, NP, DP, P, LDP1, LDP2, DR, DU, R, LDR1,
     *                  LDR2, U, LDU1, LDU2, ZERCOL, IWORK, RWORK,
     *                  TOL, IERR)
C
C     PURPOSE
C
C     To compute for a given polynomial matrix
C                                                       dp-1           dp
C         P(s) = P(0) + P(1) * s + . . . + P(dp-1) * s     + P(dp)* s   ,
C
C     a unimodular polynomial matrix U(s) such that R(s) = P(s) * U(s) is
C     column reduced.
C
C     ARGUMENTS IN
C
C        MP - INTEGER.
C            The number of rows of the polynomial matrix P(s).
C            MP >= 1.
C        NP - INTEGER.
C            The number of columns of the polynomial matrix P(s).
C            NP >= 1.
C        DP - INTEGER.
C            The degree of the polynomial matrix P(s).
C            DP >= 1.
C        P - DOUBLE PRECISION array of DIMENSION (LDP1,LDP2,DP+1).
C            The leading MP by NP by (DP+1) part of this array must contain
C            the coefficients of the polynomial matrix P(s).  Specifically,
C            P(i,j,k) must contain the coefficient of s**(k-1) of the
C            polynomial which is the (i,j)-th element of P(s), where
C            i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DP+1.
C        LDP1 - INTEGER.
C            The leading dimension of array P as declared in the calling
C            program.
C            LDP1 >= MP.
C        LDP2 - INTEGER.
C            The second dimension of array P as declared in the calling
C            program.
C            LDP2 >= NP.
C
C     ARGUMENTS OUT
C
C        DR - INTEGER.
C            The degree of the column reduced polynomial matrix R(s).
C        DU - INTEGER.
```

```
C            The degree of the unimodular polynomial matrix U(s).
C        R - DOUBLE PRECISION array of DIMENSION (LDR1,LDR2,DP+1).
C            The leading MP by NP by (DR+1) part of this array contains
C            the coefficients of the column reduced polynomial matrix R(s).
C            Specifically, R(i,j,k) contains the coefficient of s**(k-1) of
C            the polynomial which is the (i,j)-th element of R(s), where
C            i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DR+1.
C        LDR1 - INTEGER.
C            The leading dimension of array R as declared in the calling
C            program.
C            LDR1 >= MP.
C        LDR2 - INTEGER.
C            The second dimension of array R as declared in the calling
C            program.
C            LDR2 >= NP.
C        U - DOUBLE PRECISION array of DIMENSION (LDU1,LDU2,NP*DP+1).
C            The leading NP by NP by (DU+1) part of this array contains
C            the coefficients of the unimodular polynomial matrix U(s).
C            Specifically, U(i,j,k) contains the coefficient of s**(k-1) of
C            the polynomial which is the (i,j)-th element of U(s), where
C            i = 1,2,...,NP, j = 1,2,...,NP and k = 1,2,...,DU+1.
C        LDU1 - INTEGER.
C            The leading dimension of array U as declared in the calling
C            program.
C            LDU1 >= NP.
C        LDU2 - INTEGER.
C            The second dimension of array U as declared in the calling
C            program.
C            LDU2 >= NP.
C        ZERCOL - LOGICAL array of DIMENSION at least (NP).
C            If ZERCOL(j) = .TRUE., then the j-th column of R(s) is zero;
C            otherwise the j-th column belongs to R1(s) (see METHOD).
C
C     WORKSPACE
C
C        IWORK - INTEGER array of DIMENSION at least (liwork),
C            where liwork = 2*ND1*MP + NP + 1,
C            and ND1 = NP*DP + 1.
C        RWORK - DOUBLE PRECISION array of DIMENSION at least (lrwork),
C            where lrwork = (2*MP+1)*MP*ND1**2 + (2*MP*NP+4*MP+NP)*ND1 +
C                           MP*NP + 3*NP.
C
C     TOLERANCES
C
C        TOL - DOUBLE PRECISION.
C            A tolerance below which matrix elements are considered to be
C            zero. If the user sets TOL to be less than
C            EPS * (((DP+1) * MP)**2 * MAX(P(i,j,k))), then the tolerance is
```

```
C          taken as EPS * (((DP+1) * MP)**2 * MAX(P(i,j,k))), i = 1,...,MP,
C          j = 1,...,NP, k = 1,..., DP + 1, where EPS is the machine
C          precision.
C
C     ERROR INDICATOR
C
C        IERR - INTEGER.
C           Unless the routine detects an error (see next section),
C           IERR contains 0 on exit.
C
C     WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C        IERR = 1 : On entry, MP < 1 or NP < 1 or DP < 1 or
C                   LDP1 < MP or LDP2 < NP.
C        IERR = 2 : No column reduced R(s) has been found
C
C     METHOD
C
C     Let GAMC(P) be the constant matrix such that each of its columns
C     contains the coefficients of the highest power of s occurring in the
C     corresponding column of P(s), the so-called leading column coefficient
C     matrix. Then P(s) is called column reduced if there exists a permutation
C     matrix T such that P(s) = ( Z , P1(s) ) * T, where Z is a zero matrix
C     and GAMC(P1) has full column rank.
C
C     Let (U(s),Z(s))' be a minimal polynomial basis (MPB) for
C         b
C     Ker(s P(s), -I), for some b > 0. It has been proved, see [1], that if
C     b is greater than d'c(P), the sum of all but the smallest column
C                                                                    -b
C     degrees of P(s), then U(s) is unimodular and R(s) = s  Z(s) is column
C     reduced and P(s) * U(s) = R(s).
C                                                  b
C     The routine uses a linearization of (s P(s), -I) to compute an MPB
C     for b = 1,2,... and checks for each b whether R(s) is column reduced,
C     i.e. whether GAMC(R1) has full column rank. The algorithm finishes
C     with U(s) and R(s) as soon as R(s) is column reduced.
C
C     REFERENCES
C
C     [1] Neven, W.H.L. and Praagman, C.
C         Column Reduction of Polynomial Matrices.
C         Linear Algebra and its Applications 188, 189, pp.  569-589, 1993.
C
C     NUMERICAL ASPECTS
C
C
C     The algorithm used by the routine involves the construction of a
```

```
C                                                        b
C     special staircase form of a linearization of (s  P(s), -I) with
C     pivots considered to be non-zero when they are greater than or equal
C     to TOL. These pivots are then inverted in order to construct the
C                       b
C     columns of ker(s  P(s), -I).
C     The user is recommended to choose TOL of the order of the relative
C     error in the elements of P(s). If TOL is chosen to be too small, then
C     a very small element of insignificant value may be taken as pivot.
C     As a consequence, the correct null-vectors, and hence R(s), may not be
C     found. In the case that R(s) has not been found and in the case that
C     the elements of the computed U(s) and R(s) are large relative to the
C     elements of P(s) the user should consider trying several values of TOL.
C
C     CONTRIBUTORS
C
C        A.J. Geurts (Eindhoven University of Technology).
C        C. Praagman (University of Groningen).
C
C     REVISIONS
C
C        1994, February 11.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO, ONE, EPS
      PARAMETER (ZERO = 0.0D0, ONE = 1.0D0, EPS = 1.0D0/2.0D0**56)
C     .. Scalar Arguments ..
      INTEGER MP, NP, DP, LDP1, LDP2, DR, DU, LDR1, LDR2, LDU1, LDU2,
     *        IERR
      DOUBLE PRECISION TOL
C     .. Array Arguments ..
      INTEGER IWORK(*)
      DOUBLE PRECISION P(LDP1,LDP2,*), R(LDR1,LDR2,*), U(LDU1,LDU2,*),
     *                 RWORK(*)
      LOGICAL ZERCOL(*)
C     .. Local Scalars ..
      INTEGER LDA, LDAB, LDQ, LDY, LDG, MU, S, SK, A, AB, Q, Y, YI,
     *        GAMC, BMAX, DP1, K, MAMAX, NAMAX
      DOUBLE PRECISION TOLER
      LOGICAL COLRDC, PKZERO
C     .. External Subroutines/Functions ..
      EXTERNAL F06QFF, F06QGF, F06QHF, COLRD1, CKCOLR
      DOUBLE PRECISION F06QGF
      LOGICAL CKCOLR
C     .. Intrinsic Functions ..
      INTRINSIC DBLE, MAX
C
```

```
C     .. Executable Statements ..
C
C     Check the input parameters.
C
      IF (MP.LT.1 .OR. NP.LT.1 .OR. DP.LT.1 .OR. LDP1.LT.MP .OR.
     *   LDP2.LT.NP) THEN
         IERR = 1
         RETURN
      END IF
C
      IERR = 0
C
C     Computation of the tolerance. EPS is the machine precision of the
C     double precision floating-point arithmetic of a VAX computer.
C     For an other computer the value of EPS should be adapted.
C
      TOLER = ZERO
      DO 10 K = 1, DP + 1
         TOLER = MAX(TOLER, F06QGF('M', 'G', MP, NP, P(1,1,K), LDP1))
   10 CONTINUE
C
      TOLER = DBLE(((DP+1) * MP)**2) * TOLER * EPS
      IF (TOLER .LT. TOL) TOLER = TOL
C
C     Computation of the true degree of P(s).
C
      K = DP + 2
      PKZERO = .TRUE.
C     WHILE (P(k) is a zero matrix) DO
   20 IF (PKZERO .AND. (K.GT.1)) THEN
         K = K - 1
         PKZERO = (F06QGF('M','G', MP, NP, P(1,1,K), LDP1) .EQ.  ZERO)
         GO TO 20
      END IF
C     END WHILE 20
      DP1 = K - 1
C
C     Check whether P(s) is already column reduced.
C
      Q = MP + 1
      LDQ = MP
      COLRDC = CKCOLR(MP, NP, DP1, P, LDP1, LDP2, ZERCOL, RWORK(Q), LDQ,
     *                RWORK, TOLER, IERR)
      IF (COLRDC) THEN
         DR = DP1
         DO 30 K = 1, DR + 1
            CALL F06QFF('G', MP, NP, P(1,1,K), LDP1, R(1,1,K), LDR1)
   30    CONTINUE
```

```
         DU = 0
         CALL F06QHF('G', NP, NP, ZERO, ONE, U, LDU1)
         RETURN
      END IF
C
      BMAX = (NP - 1) * DP1 + 1
      MAMAX = (DP1 + BMAX) * MP
      NAMAX = MAMAX + NP
      LDA = MAMAX + 1
      LDAB = LDA
      LDY = NAMAX
      LDG = MP
      MU = 1
      S = MU + MAMAX + 1
      SK = S + MAMAX
      A = 2 * MAMAX + 1
      AB = A + (MAMAX + 1) * NAMAX
      Q = AB + (MAMAX + 1) * NAMAX
      Y = Q + MP * NP
      YI = Y + NAMAX * (NP * DP + 1)
      GAMC = YI + NP
C
      CALL COLRD1(MP, NP, DP1, P, LDP1, LDP2, DR, DU, R, LDR1, LDR2,
     *            U, LDU1, LDU2, ZERCOL, IWORK(MU), IWORK(S), IWORK(SK),
     *            RWORK(A), LDA, RWORK(AB), LDAB, RWORK(Q), LDQ,
     *            RWORK(Y), LDY, RWORK(YI), RWORK(GAMC), LDG, RWORK,
     *            TOLER, IERR)
C
C     Check whether the computed R(s) is column reduced.
C
      IF (IERR .EQ. 0) THEN
         Q = MP + 1
         COLRDC = CKCOLR(MP, NP, DR, R, LDR1, LDR2, ZERCOL, RWORK(Q),
     *                   LDQ, RWORK, TOL, IERR)
         IF (.NOT. COLRDC) IERR = 2
      ELSE
         IERR = 2
      END IF
      RETURN
C *** Last line of COLRED ***
      END
```

CKCOLR

```
      LOGICAL FUNCTION CKCOLR(MP, NP, DP, P, LDP1, LDP2, ZERCOL, Q, LDQ,
     *                        W, TOL, IERR)
C
C     PURPOSE
C
C     To check whether the polynomial matrix
C                                                        dp-1            dp
C        P(s) = P(0) + P(1) * s + . . . + P(dp-1) * s      + P(dp) * s   ,
C
C     is column reduced.
C
C     ARGUMENTS IN
C
C        MP - INTEGER.
C            The number of rows of the polynomial matrix P(s).
C            MP >= 1.
C        NP - INTEGER.
C            The number of columns of the polynomial matrix P(s).
C            NP >= 1.
C        DP - INTEGER.
C            The degree of the polynomial matrix P(s).
C            DP >= 0.
C        P - DOUBLE PRECISION array of DIMENSION (LDP1,LDP2,DP+1).
C            The leading MP by NP by (DP+1) part of this array must contain
C            the coefficients of the polynomial matrix P(s).  Specifically,
C            P(i,j,k) must contain the coefficient of s**(k-1) of the
C            polynomial which is the (i,j)-th element of P(s), where
C            i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DP+1.
C        LDP1 - INTEGER.
C            The leading dimension of array P as declared in the calling
C            program.
C            LDP1 >= MP.
C        LDP2 - INTEGER.
C            The second dimension of array P as declared in the calling
C            program.
C            LDP2 >= NP.
C
C        ARGUMENTS OUT
C
C        ZERCOL - LOGICAL array of DIMENSION at least (NP).
C            If ZERCOL(j) = .TRUE., then the j-th column of P(s) is zero;
C            otherwise the j-th column belongs to P1(s) (see METHOD).
C
C        WORKSPACE
C
```

```
C         Q - DOUBLE PRECISION array of DIMENSION (LDQ,NP).
C         LDQ - INTEGER.
C             The leading dimension of array Q as declared by the calling
C             program.
C             LDQ >= MP.
C         W - DOUBLE PRECISION array of DIMENSION (MP).
C
C      TOLERANCES
C
C         TOL - DOUBLE PRECISION.
C             A tolerance below which matrix elements are considered to be
C             zero. If the user sets TOL to be less than
C             EPS * (((DP+1) * MP)**2 * MAX(P(i,j,k))), then the tolerance
C             is taken as EPS * (((DP+1) * MP)**2 * MAX(P(i,j,k))),
C             i = 1,...,MP, j = 1,...,NP, k = 1,...,DP+1, where EPS is the
C             machine precision.
C
C      ERROR INDICATOR
C
C         IERR - INTEGER.
C            Unless the routine detects an error (see next section),
C            IERR contains 0 on exit.
C
C      WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C         IERR = 1 : Invalid input parameter(s).
C
C      METHOD
C
C      Let GAMC(P) be the constant matrix such that each of its columns
C      contains the coefficients of the highest power of s occurring in the
C      corresponding column of P(s), the so-called leading column coefficient
C      matrix. Then P(s) is called column reduced if there exists a
C      permutation matrix T such that P(s) = ( Z , P1(s) ) * T, where Z is a
C      zero matrix and GAMC(P1) has full column rank.
C
C      The algorithm used, which is in fact the QR decomposition of the
C      leading column coefficient matrix, is as follows:
C      The columns of the leading column coefficient matrix of P1(s) are
C      determined one by one, where a column is considered zero if its
C      Euclidean norm is less than TOL. To each new column the Householder
C      transformations are applied that have transformed the submatrix of the
C      former columns in upper triangular form. If the new column is
C      independent of its predecessors, then a new Householder transformation
C      is generated and applied such that the augmented matrix is upper
C      triangular.
C      The routine terminates after the last column of P(s) has been treated
C      or when the new column is dependent of its predecessors.
```

```
C
C     CONTRIBUTOR
C
C        A.J. Geurts (Eindhoven University of Technology).
C        C. Praagman (University of Groningen).
C
C     REVISIONS
C
C        1993, November 4.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO, EPS, FACTOR
      PARAMETER (ZERO = 0.0D0, EPS = 1.0D0/2.0D0**56, FACTOR = 1.0D2)
C     .. Scalar Arguments ..
      INTEGER MP, NP, DP, LDP1, LDP2, LDQ, IERR
      DOUBLE PRECISION TOL
C     .. Array Arguments ..
      DOUBLE PRECISION P(LDP1,LDP2,*), Q(LDQ,*), W(*)
      LOGICAL ZERCOL(*)
C     .. Local Scalars ..
      INTEGER H, J, J1, K
      DOUBLE PRECISION TOLER, NORM, ZETA
      LOGICAL FULLRK, NOTCJ
C     .. External Subroutines/Functions ..
      EXTERNAL DCOPY, DNRM2, F06FSF, F06FUF, F06QGF, F06QHF
      DOUBLE PRECISION DNRM2, F06QGF
C     .. Intrinsic Functions ..
      INTRINSIC DBLE, MAX, MIN
C
C     .. Executable Statements ..
C
C     Check input parameters
C
      IF (MP.LT.1 .OR. NP.LT.1 .OR. DP.LT.0 .OR. LDP1.LT.MP .OR.
     *    LDP2.LT.NP) THEN
          IERR = 1
          RETURN
      END IF
C
C     Computation of the tolerance. EPS is the machine precision of the
C     double precision floating-point arithmetic of a VAX computer.
C     For an other computer the value of EPS should be adapted.
C
      TOLER = ZERO
      DO 10 K = 1, DP + 1
         TOLER = MAX(TOLER, F06QGF('M', 'G', MP, NP, P(1,1,K), LDP1))
   10 CONTINUE
```

```
      TOLER = DBLE(((DP+1) * MP))**2 * TOLER * EPS
      IF (TOLER .LT. TOL) TOLER = TOL
C
      CALL F06QHF('G', MP, NP, ZERO, ZERO, Q, LDQ)
      J = 1
      J1 = 1
      FULLRK = .TRUE.
C     WHILE (FULLRK and J1 <= NP) DO
   20 IF (FULLRK .AND. J1.LE.NP) THEN
C
C        Find the j-th column of the leading column coefficient matrix of
C        P1(s) and put it in W.
C
         K = DP + 1
         NOTCJ = .TRUE.
C        WHILE (j-th column not found) DO
   30    IF (NOTCJ .AND. K.GE.1) THEN
            NORM = DNRM2(MP, P(1,J1,K), 1)
            IF (NORM .GE. TOLER) THEN
               CALL DCOPY(MP, P(1,J1,K), 1, W, 1)
               NOTCJ = .FALSE.
            END IF
            K = K - 1
            GO TO 30
         END IF
C        END WHILE 30
C
C        Check whether the j-th column is linearly independent of the
C        preceding columns.
C
         IF (NOTCJ) THEN
            ZERCOL(J1) = .TRUE.
            J1 = J1 + 1
         ELSE
            ZERCOL(J1) = .FALSE.
C
C           Apply the Householder transformations Qh, h = 1,...,min(mp,j) - 1,
C           to W.
C
            DO 40 H = 1, MIN(MP,J) - 1
               CALL F06FUF(MP-H, Q(H+1,H), 1, Q(H,H), W(H), W(H+1), 1)
   40       CONTINUE
            NORM = DNRM2(MP-J+1, W(J), 1)
            IF (NORM .LT. TOLER) THEN
               FULLRK = .FALSE.
            ELSE
C
```

```
C             Generate the Householder transformation Qj.
C
              IF (J .LT. MP) THEN
                 CALL F06FSF(MP-J, W(J), W(J+1), 1, TOLER, ZETA)
                 CALL DCOPY(MP-J, W(J+1), 1, Q(J+1,J), 1)
                 Q(J,J) = ZETA
              ELSE
                 Q(J,J) = ZERO
              END IF
           END IF
           J = J + 1
           J1 = J1 + 1
        END IF
        GO TO 20
     END IF
C    END WHILE 20
     CKCOLR = FULLRK
     RETURN
C *** Last line of CKCOLR ***
     END
```

CKGAMC

```
      LOGICAL FUNCTION CKGAMC(MP, INV, GAMC, LDG, Q, LDQ, RWORK, TOL)
C
C     PURPOSE
C
C     To check whether the leading coefficient matrix has still full column
C     rank after a new column has been appended.
C     REMARK: This auxiliary routine is intended to be called only from the
C             routine COLRED.
C
C     ARGUMENTS IN
C
C        MP - INTEGER.
C            The number of rows of the matrix GAMC.
C            MP >= 1.
C        INV - INTEGER.
C            The number of columns of the matrix GAMC.
C            INV >= 1.
C        GAMC - DOUBLE PRECISION array of DIMENSION (LDG,INV)
C            The leading MP by INV part of this array must contain the
C            leading coefficient matrix GAMC of which the first INV - 1
C            columns have been transformed in upper triangular form.
C            Note: this array is overwritten
C        LDG - INTEGER.
```

```
C             The leading dimension of array GAMC as declared in the calling
C             program.
C             LDG >= MP.
C         Q - DOUBLE PRECISION array of DIMENSION (LDQ,INV)
C             The leading MP by INV - 1 part of this array must contain the
C             vectors of the elementary Householder transformations by which
C             the first INV - 1 columns of GAMC have been transformed into an
C             upper triangular matrix.
C             Note; this array is overwritten.
C         LDQ - INTEGER.
C             The leading dimension of array Q as declared in the calling
C             program.
C             LDQ >= MP.
C
C      ARGUMENTS OUT
C
C         GAMC - DOUBLE PRECISION array of DIMENSION (LDG,INV)
C             The leading MP by INV part of this array contains the leading
C             coefficient matrix GAMC transformed in upper triangular form.
C         Q - DOUBLE PRECISION array of DIMENSION (LDQ,INV)
C             The leading MP by INV part of this array contains the vectors
C             of the elementary Householder transformations by which GAMC has
C             been transformed into an upper triangular matrix.
C
C         WORKSPACE
C
C         RWORK - DOUBLE PRECISION array of DIMENSION (MP)
C
C      TOLERANCES
C
C         TOL - DOUBLE PRECISION.
C             A tolerance below which matrix elements are considered to be
C             zero.
C
C      METHOD
C
C      Let the first INV - 1 columns of GAMC be linearly independent, which
C      has been checked by former calls of CKGAMC. A new column is appended.
C      To this column the Householder transformations are applied that have
C      transformed the matrix of the former columns in upper triangular form.
C      If the new column is independent of its predecessors, then a new
C      Householder transformation is generated and applied such that the
C      augmented matrix is upper triangular.
C
C      CONTRIBUTOR
C
C         A.J. Geurts.
C
```

```
C     REVISIONS
C
C        1993, October 29.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO = 0.0D0)
C     .. Scalar Arguments ..
      INTEGER MP, INV, LDG, LDQ
      DOUBLE PRECISION TOL
C     .. Array Arguments ..
      DOUBLE PRECISION GAMC(LDG,*), Q(LDQ,*), RWORK(*)
C     .. Local Scalars ..
      INTEGER J
      DOUBLE PRECISION NORM, ZETA
C     .. External Subroutines/Functions ..
      EXTERNAL DCOPY, DNRM2, F06FBF, F06FSF, F06FUF
      DOUBLE PRECISION DNRM2
C
C     .. Executable Statements ..
C
      IF (INV .LE. MP) THEN
         CALL F06FBF(MP, ZERO, Q(1,INV), 1)
C
C        Check whether the INV-th column is linearly independent of the
C        preceding columns by applying the Householder transformations Q(h),
C        h = 1,...,INV-1, to the INV-th column.
C
         CALL DCOPY(MP, GAMC(1,INV), 1, RWORK, 1)
         DO 10 J = 1, INV - 1
            CALL F06FUF(MP-J, Q(J+1,J), 1, Q(J,J), RWORK(J), RWORK(J+1),
     *                  1)
   10    CONTINUE
         NORM = DNRM2(MP-INV+1, RWORK(INV), 1)
         IF (NORM .LT. TOL) THEN
            CKGAMC = .FALSE.
         ELSE
C
C           Generate the Householder transformation Q(INV) if INV < MP.
C
            CALL DCOPY(MP, RWORK, 1, GAMC(1,INV), 1)
            IF (INV .LT. MP) THEN
               CALL F06FSF(MP-INV, RWORK(INV), RWORK(INV+1), 1, TOL,
     *                     ZETA)
               GAMC(INV,INV) = RWORK(INV)
               CALL F06FBF(MP-INV, ZERO, GAMC(INV+1,INV), 1)
               CALL DCOPY(MP-INV, RWORK(INV+1), 1, Q(INV+1,INV), 1)
```

```
            Q(INV,INV) = ZETA
         ELSE
            Q(INV,INV) = ZERO
         END IF
         CKGAMC = .TRUE.
      END IF
   ELSE
      CKGAMC = .FALSE.
   END IF
   RETURN
C *** Last line of CKGAMC ***
   END
```

COLRD1

```
      SUBROUTINE COLRD1(MP, NP, DP, P, LDP1, LDP2, DR, DU, R, LDR1,
     *                  LDR2, U, LDU1, LDU2, ZERCOL, MU, S, SK, A, LDA,
     *                  AB, LDAB, Q, LDQ, Y, LDY, YI, GAMC, LDG, W,
     *                  TOL, IERR)
C
C     PURPOSE
C
C     To compute for a given polynomial matrix
C                                                   dp-1           dp
C        P(s) = P(0) + P(1) * s + . . . + P(dp-1) * s     + P(dp) * s   ,
C
C     which is not column reduced, a unimodular polynomial matrix U(s) such
C     that R(s) = P(s) * U(s) is column reduced.
C     REMARK: This auxiliary routine is intended to be called only from the
C             routine COLRED.
C
C     ARGUMENTS IN
C
C        MP - INTEGER.
C             The number of rows of the polynomial matrix P(s).
C             MP >= 1.
C        NP - INTEGER.
C             The number of columns of the polynomial matrix P(s).
C             NP >= 1.
C        DP - INTEGER.
C             The degree of the polynomial matrix P(s).
C             DP >= 1.
C        P - DOUBLE PRECISION array of DIMENSION (LDP1,LDP2,DP+1).
C             The leading MP by NP by (DP+1) part of this array must contain
C             the coefficients of the polynomial matrix P(s).  Specifically,
C             P(i,j,k) must contain the coefficient of s**(k-1) of the
```

```
C            polynomial which is the (i,j)-th element of P(s), where
C            i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DP+1.
C        LDP1 - INTEGER.
C            The leading dimension of array P as declared in the calling
C            program.
C            LDP1 >= MP.
C        LDP2 - INTEGER.
C            The second dimension of array P as declared in the calling
C            program.
C            LDP2 >= NP.
C
C     ARGUMENTS OUT
C
C        DR - INTEGER.
C            The degree of the column reduced polynomial matrix R(s).
C        DU - INTEGER.
C            The degree of the unimodular polynomial matrix U(s).
C        R - DOUBLE PRECISION array of DIMENSION (LDR1,LDR2,DP+1).
C            The leading MP by NP by (DR+1) part of this array contains
C            the coefficients of the column reduced polynomial matrix R(s).
C            Specifically, R(i,j,k) contains the coefficient of s**(k-1) of
C            the polynomial which is the (i,j)-th element of R(s), where
C            i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DR+1.
C        LDR1 - INTEGER.
C            The leading dimension of array R as declared in the calling
C            program.
C            LDR1 >= MP.
C        LDR2 - INTEGER.
C            The second dimension of array R as declared in the calling
C            program.
C            LDR2 >= NP.
C        U - DOUBLE PRECISION array of DIMENSION (LDU1,LDU2,NP*DP+1).
C            The leading NP by NP by (DU+1) part of this array contains
C            the coefficients of the unimodular polynomial matrix U(s).
C            Specifically, U(i,j,k) contains the coefficient of s**(k-1) of
C            the polynomial which is the (i,j)-th element of U(s), where
C            i = 1,2,...,NP, j = 1,2,...,NP and k = 1,2,...,DU+1.
C        LDU1 - INTEGER.
C            The leading dimension of array U as declared in the calling
C            program.
C            LDU1 >= NP.
C        LDU2 - INTEGER.
C            The second dimension of array U as declared in the calling
C            program.
C            LDU2 >= NP.
C        ZERCOL - LOGICAL array of DIMENSION at least (NP).
C            If ZERCOL(j) = .TRUE., then the j-th column of R(s) is zero;
C            otherwise the j-th column belongs to R1(s) (see METHOD).
```

```
C
C       WORKSPACE
C
C       MU - INTEGER array of DIMENSION at least (mamax+1),
C           where mamax = (NP * DP + 1) * MP.
C           On exit, this array contains the row indices of the left upper
C           elements of the right invertible diagonal submatrices of A'.
C       S - INTEGER array of DIMENSION at least (mamax).
C           On exit, this array contains the column indices of the pivots
C           of A', that is A transformed in upper staircase form.
C       SK - INTEGER array of DIMENSION at least (NP).
C       A - DOUBLE PRECISION array of DIMENSION (LDA,namax),
C           where namax = mamax + NP.
C           On exit, the upper block diagonal part of this array contains
C           the transformed matrix A', the lower part contains the vectors
C           of the Householder transformations.
C       LDA - INTEGER.
C           The leading dimension of array A as declared in the calling
C           program.
C           LDA >= mamax + 1.
C       AB - DOUBLE PRECISION array of DIMENSION (LDAB,namax).
C           Array in which the transformed matrix Ab is saved.
C       LDAB - INTEGER.
C           The leading dimension of array AB as declared in the calling
C           program.
C           LDAB >= mamax + 1.
C       Q - DOUBLE PRECISION array of DIMENSION (LDQ,NP).
C       LDQ - INTEGER.
C           The leading dimension of array Q as declared in the calling
C           program.
C           LDQ >= MP.
C       Y - DOUBLE PRECISION array of DIMENSION (LDY,NP*DP+1).
C           Array in which a null vector of sA - E or sA' - E is stored.
C       LDY - INTEGER.
C           The leading dimension of array Y as declared in the calling
C           program.
C           LDA >= namax.
C       YI - DOUBLE PRECISION array of DIMENSION at least (NP).
C       GAMC - DOUBLE PRECISION array of DIMENSION (LDG,NP)
C           On exit, this array contains the leading column coefficient
C           matrix of R(s), which has full column rank.
C       LDG - INTEGER.
C           The leading dimension of array GAMC as declared in the calling
C           program.
C           LDG >= MP.
C       W - DOUBLE PRECISION array of DIMENSION (2*mamax).
C
C     TOLERANCES
```

```
C
C        TOL - DOUBLE PRECISION.
C            A tolerance below which matrix elements are considered to be
C            zero.
C
C     ERROR INDICATOR
C
C        IERR - INTEGER.
C           Unless the routine detects an error (see next section),
C           IERR contains 0 on exit.
C
C     WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C        IERR = 2 : No column reduced R(s) has been found for the maximum b,
C                   (see METHOD).
C        IERR = 3 : The computation of a null vector has failed, because a
C                   diagonal block of A' is not right invertible.
C        IERR = 4 : The computation of R(s) has failed, because a computed
C                   null vector is not s**B times a polynomial vector.
C        IERR = 5 : The degree of R(s) has become greater than the degree of
C                   P(s).
C
C     METHOD
C
C     Let GAMC(P) be the constant matrix such that each of its columns
C     contains the coefficients of the highest power of s occurring in the
C     corresponding column of P(s), the so-called leading column coefficient
C     matrix. Then P(s) is called column reduced if there exists a permutation
C     matrix T such that P(s) = ( Z , P1(s) ) * T, where Z is a zero matrix
C     and GAMC(P1) has full column rank.
C
C     Let (U(s),Z(s))' be a minimal polynomial basis (MPB) for
C           b
C     Ker(s P(s), -I), for some b > 0. It has been proved, see [1], that if
C     b is greater than d'c(P), the sum of all but the smallest column
C                                                                 -b
C     degrees of P(s), then U(s) is unimodular and R(s) = s  Z(s) is column
C     reduced and P(s) * U(s) = R(s).
C     The routine computes an MPB for b = 1,2,... and checks for each b
C     whether R(s) is column reduced, i.e. whether GAMC(R1) has full column
C     rank. The algorithm finishes with U(s) and R(s) as soon as R(s) is
C     column reduced.
C
C     REFERENCES
C
C     [1] Neven, W.H.L. and Praagman, C.
C         Column Reduction of Polynomial Matrices.
C         Linear Algebra and its Applications 188, 189, pp.  569-589, 1993.
```

```
C
C     CONTRIBUTORS
C
C        A.J. Geurts (Eindhoven University of Technology).
C        C. Praagman (University of Groningen).
C
C     REVISIONS
C
C        1993, November 8.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER (ZERO = 0.0D0, ONE = 1.0D0)
C     .. Scalar Arguments ..
      INTEGER MP, NP, DP, LDP1, LDP2, DR, DU, LDR1, LDR2, LDU1, LDU2,
     *        LDA, LDAB, LDQ, LDY, LDG, IERR
      DOUBLE PRECISION TOL
C     .. Array Arguments ..
      INTEGER MU(*), S(*), SK(*)
      DOUBLE PRECISION P(LDP1,LDP2,*), R(LDR1,LDR2,*), U(LDU1,LDU2,*),
     *                 A(LDA,*), AB(LDAB,*), Q(LDQ,*), Y(LDY,*), YI(*),
     *                 GAMC(LDG,*),  W(*)
      LOGICAL ZERCOL(*)
C     .. Local Scalars ..
      INTEGER BMAX, B, I, IC, IR, IND, NNV, NNVB, NCR1, J, K, KK,
     *        L, C1AI, DUB, DUR1, MA, NA, MAI, NAI, MUI
      DOUBLE PRECISION MUQ, NORM, ZETA
      LOGICAL COLRDC
C     .. External Subroutines/Functions ..
      EXTERNAL DCOPY, DGEMV, DGER, DNRM2, F06FBF, F06QFF, F06QHF,
     *         COMPYI, CKGAMC, COMPTY, COMPTV, HHTRAN, MKPENC
      DOUBLE PRECISION DNRM2
      LOGICAL CKGAMC
C     .. Intrinsic Functions ..
      INTRINSIC MAX
C
C     .. Executable Statements ..
C
      BMAX = (NP-1) * DP + 1
      CALL MKPENC(MP, NP, DP, P, LDP1, LDP2, AB, LDAB, MA, NA)
      NNVB = 0
      DUB = -1
      DO 10 K = 1, NP * DP + 1
         CALL F06QHF('G', NP, NP, ZERO, ZERO, U(1,1,K), LDU1)
   10 CONTINUE
      COLRDC = .FALSE.
      MU(1) = 1
```

```
      B = 0
C     WHILE (.NOT.COLRDC and B < BMAX) DO
   20 IF (.NOT.COLRDC .AND. B.LT.BMAX) THEN
         B = B + 1
C
C        Initialization of A.
C
         IF (B .EQ. 1) THEN
            CALL F06QFF('G', MA, NA, AB, LDAB, A, LDA)
            CALL F06FBF(NA, ZERO, A(MA+1,1), LDA)
         ELSE
            CALL F06QFF('G', MA+1, NA, AB, LDAB, A, LDA)
            J = NP + MU(B-1)
            L = MA - MP - MU(B-1) + 1
            CALL F06QHF('G', MP, J-1, ZERO, ZERO, A(MA+2,1), LDA)
            CALL F06QHF('G', MP+1, L, ZERO, ZERO, A(MA+1,J), LDA)
            CALL F06QHF('G', MP+1, MP, ZERO, ONE, A(MA+1,L+J), LDA)
            MA = MA + MP
            CALL F06QHF('G', MA+1, MP, ZERO, ZERO, A(1,NA+1), LDA)
            NA = NA + MP
            DO 30 K = MU(B-1), MU(B) - 1
               ZETA = A(K+1,S(K))
               IF (ZETA .NE. ZERO) THEN
                  J = NP + K
                  L = MA - MP - K + 1
                  MUQ = ONE/ZETA
                  CALL DGEMV('N', MP, L, ONE, A(MA-MP+1,J), LDA,
     *                        A(K+1,S(K)), 1, ZERO, W, 1)
                  CALL DGER(MP, L, -MUQ, W, 1, A(K+1,S(K)), 1,
     *                      A(MA-MP+1,J), LDA)
               END IF
   30       CONTINUE
         END IF
C
         I = B
         DU = DUB
         NNV = NNVB
         DR = -1
         DO 40 K = 1, DU + 1
            CALL F06QHF('G', NP, NP-NNVB, ZERO, ZERO, U(1,NNVB+1,K),
     *                  LDU1)
   40    CONTINUE
         DO 50 K = 1, DP + 1
            CALL F06QHF('G', MP, NP, ZERO, ZERO, R(1,1,K), LDR1)
   50    CONTINUE
         NCR1 = 0
         COLRDC = .TRUE.
C
```

```
C        NNV is the number of already found null vectors. With these
C        null vectors correspond the NP by NNV column reduced matrix R(s).
C        NCR1 is the number of columns in R1(s).
C
C        WHILE (NNV < NP, which means that not all null vectors of sA - E
C               have been found, and R(s) is column reduced) DO
   60    IF ((NNV.LT.NP) .AND. COLRDC) THEN
C
C           Determine A(i,i), the i(th) right invertible diagonal block.
C           The left upper element of A(i,i) is A(MUI,C1AI) and the row and
C           column dimensions are MAI and NAI, respectively.
C
            MUI = MU(I)
            MAI = 0
            IF (I .EQ. 1) THEN
               C1AI = 1
               NAI = NP
            ELSE
               C1AI = MU(I-1) + NP
               NAI = MU(I) - MU(I-1)
            END IF
C
C           Compute the null vectors of A(i,i) and the corresponding null-
C           vectors of sA - E one by one, append the appropriate parts of a
C           computed null vector to U(s) and R(s) and check whether R(s)
C           remains column reduced.
C
            K = 1
C           WHILE (K <= NAI and R(s) still column reduced) DO
   70       IF (K.LE.NAI .AND. COLRDC) THEN
               IR = MUI + MAI
               IC = C1AI + K - 1
               L = MA - IR + 1
               IF (L .GT. 0) THEN
                  NORM = DNRM2(L, A(IR,IC), 1)
               ELSE
                  NORM = ZERO
               END IF
               IF (NORM .GE. TOL) THEN
C
C                 Generate and apply the Householder transformation for the
C                 IC-th column of A
C
                  CALL HHTRAN(MA, NA, IC, IR, A, LDA, W, W(MA+1), TOL)
                  CALL DCOPY(L, W, 1, A(IR+1,IC), 1)
                  S(MUI+MAI) = IC
                  MAI = MAI + 1
                  SK(MAI) = IC - C1AI + 1
```

```
            ELSE
C
C              Compute the null vector Y of sA - E corresponding
C              to the IC-th column of A.
C
               IF (L .GT. 0) THEN
                  CALL F06FBF(L, ZERO, A(IR,IC), 1)
               END IF
               IF (MAI .GT. 0) THEN
                  CALL DCOPY(MAI, A(MUI,IC), 1, W, 1)
                  CALL COMPYI(MAI, K-1, A(MUI,C1AI), LDA, SK, W,
     *                        YI, IERR)
                  IF (IERR .NE. 0) RETURN
               ELSE
                  CALL F06FBF(K-1, ZERO, YI, 1)
               END IF
               YI(K) = -ONE
               CALL COMPTY(NP, NA, I, K, YI, MU, A, LDA, S, Y, LDY,
     *                     SK, W, IERR)
               IF (IERR .NE. 0) THEN
                  IERR = 3
                  RETURN
               END IF
               CALL COMPTV(MA, NA, I, IR, A, LDA, S, Y, LDY, W)
C
C              Append the first NP by I block of Y to the unimodular U,
C              and the last MP by I block to the column reduced R and the
C              last MP elements of the I-th column of Y to the leading
C              column coefficient matrix GAMC.
C
               NNV = NNV + 1
               DUR1 = 0
               DO 80 KK = 1, I
                  NORM = DNRM2(NP, Y(1,KK), 1)
                  IF (NORM .GE. TOL) THEN
                     DUR1 = KK - 1
                     CALL DCOPY(NP, Y(1,KK), 1, U(1,NNV,KK), 1)
                  ELSE
                     CALL F06FBF(NP, ZERO, U(1,NNV,KK), 1)
                  END IF
   80          CONTINUE
               DU = MAX(DU, DUR1)
               IND = NA - MP + 1
               DO 90 KK = 1, B
                  NORM = DNRM2(MP, Y(IND,KK), 1)
                  IF (NORM .NE. ZERO) THEN
                     IERR = 4
                     RETURN
```

```
                  END IF
   90          CONTINUE
               DUR1 = -1
               DO 100 KK = B + 1, I
                  NORM = DNRM2(MP, Y(IND,KK), 1)
                  IF (NORM .GE. TOL) THEN
                     DUR1 = KK - B - 1
                     CALL DCOPY(MP, Y(IND,KK), 1, R(1,NNV,KK-B), 1)
                  ELSE
                     CALL F06FBF(MP, ZERO, R(1,NNV,KK-B), 1)
                  END IF
  100          CONTINUE
               IF (DUR1 .NE. -1) THEN
                  NCR1 = NCR1 + 1
                  ZERCOL(NNV) = .FALSE.
                  DR = MAX(DR, DUR1)
                  IF (DR .GT. DP) THEN
                     IERR = 5
                     RETURN
                  END IF
                  CALL DCOPY(MP, Y(IND,DUR1+B+1), 1, GAMC(1,NCR1), 1)
C
C                 Check whether R is still column reduced.
C
                  COLRDC = CKGAMC(MP, NCR1, GAMC, LDG, Q, LDQ, W,
     *                            TOL)
               ELSE
                  ZERCOL(NNV) = .TRUE.
               END IF
            END IF
            K = K + 1
            GO TO 70
         END IF
C        END WHILE 70
C
C        Save the transformed A if I = B.
C
         IF (I .EQ. B) THEN
            CALL F06QFF('G', MA+1, NA, A, LDA, AB, LDAB)
            NNVB = NNV
            DUB = DU
         END IF
C
         IF (NNV.LE.NP .AND. COLRDC) THEN
            I = I + 1
            MU(I) = MU(I-1) + MAI
         END IF
         GO TO 60
```

```
         END IF
C        END WHILE 60
         GO TO 20
      END IF
C     END WHILE 20
C
      IF (.NOT. COLRDC) IERR = 2
      RETURN
C *** Last line of COLRD1 ***
      END
```

COMPTV

```
      SUBROUTINE COMPTV(MA, NA, I, J, A, LDA, S, Y, LDY, RWORK)
C
C     PURPOSE
C
C     To apply the Householder reflections, thus far applied to sA - E, to
C     the null vector Y(s) corresponding to a given column of the transformed
C     sA' - E, which transforms this null vector into a null vector V(s) of
C     the original pencil sA - E.
C     REMARK: This auxiliary routine is intended to be called only from the
C             routine COLRED.
C
C     ARGUMENTS IN
C
C        MA - INTEGER.
C            The number of rows of matrix A.
C            MA >= 1.
C        NA - INTEGER.
C            The number of columns of matrix A.
C            NA >= MA.
C        I - INTEGER.
C            The actual number of columns in Y, which is also the index of
C            the diagonal block in A corresponding to the null vector Y(s).
C            I >= 1.
C        J - INTEGER.
C            The row index in A which corresponds to the null vector Y(s).
C            J >= 1.
C        A - DOUBLE PRECISION array of DIMENSION (LDA,NA).
C            The leading (MA + 1) by NA part of this array must contain the
C            transformed matrix A' in the upper part and the Householder
C            transformation vectors in the lower part.
C        LDA - INTEGER.
C            The leading dimension of array A as declared in the calling
C            program.
```

```
C            LDA >= MA + 1.
C        S - INTEGER ARRAY of DIMENSION at least (J-1).
C            The leading J - 1 elements of this array must contain the
C            indices of the pivots of the right invertible diagonal sub-
C            matrices, i.e., the pivot of A(m,m) is A(m,S(m)), M=1,...,J-1.
C            S(m) is also the index of the column in array A in which the
C            m-th non-trivial Householder transformation vector is stored.
C        Y - DOUBLE PRECISION array of DIMENSION (LDY,I).
C            The leading NA by I part of this array must contain the
C            polynomial null vector Y(s) of sA' - E to be transformed, where
C            the t-th column (t = 1,...,I) must contain the coefficient of
C            s**(t-1).
C            Note: this array is overwritten.
C        LDY - INTEGER.
c            The leading dimension of array Y as declared in the calling
C            program.
C            LDY >= NA.
C
C     ARGUMENTS OUT
C
C        Y - DOUBLE PRECISION array of DIMENSION (LDY,I).
C            The leading NA by I part of this array contains the transformed
C            null vector V(s) = Q * Y(s), where Q  is the product of the
C            (J-1) Householder transformations Q(m).
C
C     WORKSPACE
C
C        RWORK - DOUBLE PRECISION array of DIMENSION (2*MA).
C
C     METHOD
C
C     Let
C         Q(m)  = ( I    0  )
C                 ( 0  P(m) )
C     be the elementary Householder transformation corresponding to the
C     pivot A(m,S(m)), augmented such that Q(m) is NA by NA, then
C     Q(1) Q(2) ... Q(J-1) Y ==> Y is computed.
C
C     CONTRIBUTOR
C
C        A.J. Geurts.
C
C     REVISIONS
C
C        1992, October 27.
C
      IMPLICIT NONE
C     .. Parameters ..
```

```
      DOUBLE PRECISION ZERO, ONE
      PARAMETER (ZERO = 0.0D0, ONE = 1.0D0)
C     .. Scalar Arguments ..
      INTEGER MA, NA, I, J, LDA, LDY
C     .. Array Arguments ..
      INTEGER S(*)
      DOUBLE PRECISION A(LDA,NA), Y(LDY,I), RWORK(2*NA)
C     .. Local Scalars ..
      INTEGER LEN, M, M1, MY
      DOUBLE PRECISION NU
C     .. External Subroutines/Functions ..
      EXTERNAL DCOPY, DGER, DGEMV, F06FBF
C
C     .. Executable Statements ..
C
C     Compute the matrix P(m) Y by w = Y'u and next Y = Y - nu * uw'.
C     for m = j-1, ... , 1.
C
      DO 20 M = J - 1, 1, -1
         IF (A(M+1,S(M)) .NE. ZERO) THEN
            LEN = MA - M + 1
            MY = NA - MA + M
            M1 = MA + 1
            CALL DCOPY(LEN, A(M+1,S(M)), 1, RWORK, 1)
            CALL F06FBF(LEN, ZERO, RWORK(M1), 1)
            NU = ONE/RWORK(1)
            CALL DGEMV('T', LEN, I, ONE, Y(MY,1), LDY, RWORK, 1,
     *                 ZERO, RWORK(M1), 1)
            CALL DGER(LEN, I, -NU, RWORK, 1, RWORK(M1), 1, Y(MY,1), LDY)
         END IF
   20 CONTINUE
      RETURN
C *** Last line of COMPTV ***
      END
```

COMPTY

```
      SUBROUTINE COMPTY(NP, NA, I, NYI, YI, MU, A, LDA, S, Y, LDY, SK,
     *                  RWORK, IERR)
C
C     PURPOSE
C
C     To compute a right null vector of the pencil sA' - E, where the left
C     part of A' is in staircase form. Actually, the computed vector is the
C     null vector of the corresponding left part of the pencil.
C     REMARK: This auxiliary routine is intended to be called only from the
```

```
C             routine COLRED.
C
C     ARGUMENTS IN
C
C        NP - INTEGER.
C            The number of columns of the polynomial matrix.
C            NP >= 1.
C        NA - INTEGER.
C            The number of columns of the matrix A.
C            NA >= 1.
C        I - INTEGER.
C            The index of the current diagonal block A(i,i) of the matrix A
C            being transformed into staircase form.
C            I >= 1.
C        NYI - INTEGER.
C            The length of the righthandside vector YI.
C            1 <= NYI <= NP.
C        YI - DOUBLE PRECISION array of DIMENSION at least (NP).
C            The right null vector of A(i,i).
C        MU - INTEGER array of DIMENSION at least (MA).
C            MU(k), k = 1, ..., i must contain the row index of the left
C            upper element of A(k,k).
C        A - DOUBLE PRECISION array of DIMENSION (LDA,NA).
C            The leading MU(i) - 1 by MU(i) + NP - 1 part of this array must
C            contain the part of the matrix A' which is in staircase form.
C        LDA - INTEGER.
C            The leading dimension of array A as declared in the calling
C            program.
C            LDA >= MU(i) - 1.
C        S - INTEGER array of DIMENSION at least (MA).
C            The leading MU(i) - 1 elements of this array must contain the
C            column indices of the pivots of the right invertible diagonal
C            matrices A(k,k), k = 1, ..., i-1.
C
C     ARGUMENTS OUT
C
C        Y - DOUBLE PRECISION array of DIMENSION (LDY,NA).
C            The leading NA by i part of this array contains the computed
C            polynomial right null vector Y(s) of sA' - E, where the j-th
C            column contains the coefficient of s**(j-1).
C            The last NA - MU(i) - NP + 1 components of Y(s) are zero.
C        LDY - INTEGER.
C            The leading dimension of array Y as declared in the calling
C            program.
C            LDY >= NA.
C
C     WORK SPACE
C
```

```
C          SK - INTEGER array of DIMENSION at least (NP).
C          RWORK - DOUBLE PRECISION array of DIMENSION at least (NP).
C
C       ERROR INDICATOR
C
C          IERR - INTEGER.
C             Unless the routine detects an error (see next section),
C             IERR contains 0 on exit.
C
C       WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C       IERR = k : A(k,k) is not right invertible.
C
C       METHOD
C
C       Let the pencil sA' - E, partially transformed up to the i-th block, be
C
C          (  sA(1,1) sA(1,2)-E(1) sA(1,3)       . . .  sA(1,i)  . . .    )
C          (          sA(2,2)      sA(2,3)-E(2)  . . .  sA(2,i)  . . .    )
C          (                          .                    .              )
C          (                                 .             .              )
C          (                                        .      .              )
C          (                                            sA(i,i)  . . .    )
C          (                                                     . . .    )
C          (                                                     . . .    )
C
C       where A(k,k), k = 1,..., i is right invertible, A(k,k+1) is square and
C       E(k) = I of appropriate size.
C       Let Y(i,i), the (constant) right null vector of A(i,i), be given.
C       Then the routine computes a right null vector of sA' - E of the form
C
C          ( Y(1,1) .      Y(1,k) . . . Y(1,j) . . Y(1,i)    )
C          (          .                                      )
C          (             .                                   )
C          (               Y(k,k) . . . Y(k,j) . . Y(k,i)    )
C          (                  .                              )
C          (                         .                       )
C          (                                .      Y(i-1,i)  )
C          (                                       Y(i,i)    )
C          (   0                                     0       )
C
C       by comparing coefficients of equal power of s in the formula
C                  i             j                  k
C         A(k,k) SUM Y(k,j) * s   = Y(k+1,k+1) * s   +
C                j=k
C          i-1                  j                      j     i                    i
C          SUM (Y(k+1,j+1) - SUM A(k,l) Y(l,j)) * s    - SUM A(k,l) Y(l,i) * s
C         j=k+1                l=k+1                        l=k+1
```

```
C
C     Y(k,j), j = k,..., i, is a vector of length MU(k) - MU(k-1).
C
C     CONTRIBUTOR
C
C        A.J. Geurts.
C
C     REVISIONS
C
C        1992, October 27.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER (ZERO = 0.0D0, ONE = 1.0D0)
C     .. Scalar Arguments ..
      INTEGER NP, NA, I, NYI, LDA, LDY, IERR
C     .. Array Arguments ..
      INTEGER MU(*), S(*), SK(*)
      DOUBLE PRECISION YI(*), A(LDA,*), Y(LDY,*), RWORK(*)
C     .. Local Scalars ..
      INTEGER J, K, M, INDEXK, MUK, MUK1, NUK, MUK1NP
C     .. External Subroutines ..
      EXTERNAL DCOPY, DGEMV, F06FBF, F06QHF, COMPYI
C
C     .. Executable Statements ..
C
      IERR = 0
C
C     Initialization of the polynomial null vector Y(s).
C
      CALL F06QHF('G', NA, I, ZERO, ZERO, Y, LDY)
      K = I
      IF (I .EQ. 1) THEN
         CALL DCOPY(NYI, YI, 1, Y(1,1), 1)
      ELSE
         INDEXK = NP + MU(I-1)
         CALL DCOPY(NYI, YI, 1, Y(INDEXK,I), 1)
         DO 30 K = I - 1, 1, -1
            MUK = MU(K)
            INDEXK = NP + MUK
            NUK = MU(K+1) - MUK
            DO 20 J = K, I
C
C              Compute the righthandside for Y(k,j) and store the
C              result in RWORK.
C
               IF (J .LT. I) THEN
```

```
                  CALL DCOPY(NUK, Y(INDEXK,J+1), 1, RWORK, 1)
               ELSE
                  CALL F06FBF(NUK, ZERO, RWORK, 1)
               END IF
               IF (J .GT. K) THEN
                  CALL DGEMV('N', NUK, MU(J)-MUK, -ONE, A(MUK,INDEXK),
     *                       LDA, Y(INDEXK,J), 1, ONE, RWORK, 1)
               END IF
C
C              Solve A(k,k) * Y(k,j) = RWORK for Y(k,j).
C
               IF (K .EQ. 1) THEN
                  CALL COMPYI(NUK, NP, A(1,1), LDA, S, RWORK, Y(1,J),
     *                       IERR)
               ELSE
                  MUK1 = MU(K-1)
                  MUK1NP = MUK1 + NP
                  DO 10 M = 1, NUK
                     SK(M) = S(MUK-1+M) - MUK1NP + 1
   10             CONTINUE
                  CALL COMPYI(NUK, MUK-MUK1, A(MUK,MUK1NP), LDA, SK,
     *                       RWORK, Y(MUK1NP,J), IERR)
               END IF
               IF (IERR .NE. 0) THEN
                  IERR = K
                  RETURN
               END IF
   20       CONTINUE
   30    CONTINUE
      END IF
      RETURN
C *** Last line of COMPTY ***
      END
```

COMPYI

```
      SUBROUTINE COMPYI(M, N, A, LDA, S, V, Y, IERR)
C
C     PURPOSE
C
C     To compute a null vector yi of the right invertible diagonal submatrix
C     A(i,i), by solving an appropriate M by N system of linear equations
C     A y = v, where A is in staircase form.
C     REMARK: This auxiliary routine is intended to be called only from the
C             routine COLRED.
C
```

```
C      ARGUMENTS IN
C
C      M - INTEGER.
C          The number of rows of matrix A.
C          M >= 1.
C      N - INTEGER.
C          The number of columns of matrix A.
C          N >= M.
C      A - DOUBLE PRECISION array of DIMENSION (LDA,N).
C          The leading M by N part of this array must contain the matrix A.
C      LDA - INTEGER.
C          The leading dimension of array A as declared by the calling
C          program.
C          LDA >= M.
C      S - INTEGER array of DIMENSION at least (M).
C          S(i), i = 1,...,M, must contain the column index of the corner
C          in the i-th row of A.
C      V - DOUBLE PRECISION array of DIMENSION at least (M).
C          The righthand-side of the system of linear equations.
C
C      ARGUMENTS OUT
C
C      Y - DOUBLE PRECISION array of DIMENSION at least (N).
C          The computed solution of the system of linear equation.
C
C      ERROR INDICATOR
C
C      IERR - INTEGER.
C          Unless the routine detects an error (see next section),
C          IERR contains 0 on exit.
C
C      WARNINGS AMD ERRORS DETECTED BY THE ROUTINE
C
C      IERR = 3 : The matrix A is not right invertible.
C
C      METHOD
C
C      Let A * P = ( B | Z ) where P is a permutation matrix such that B is
C      nonsingular upper triangular. Z contains the remaining columns of A.
C      Then the system B x = v is solved and y = P ( x | 0 )'.
C
C      CONTRIBUTOR
C
C         A.J. Geurts.
C
C      REVISIONS
C
C         1992, October 27.
```

```
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO = 0.0D0)
C     .. Scalar Arguments ..
      INTEGER M, N, LDA, IERR
C     .. Array Arguments ..
      INTEGER S(M)
      DOUBLE PRECISION A(LDA,N), V(M), Y(N)
C     .. Local Scalars ..
      INTEGER J, K, SI
      DOUBLE PRECISION SUM
      LOGICAL FAIL
C     .. External Subroutines ..
      EXTERNAL F06BLF, F06FBF
      DOUBLE PRECISION F06BLF
C
C     .. Executable Statements ..
C
C     Check input parameters.
C
      IF (N .LT. M) THEN
         IERR = 3
         RETURN
      END IF
C
      IERR = 0
      CALL F06FBF(N, ZERO, Y, 1)
      SI = S(M)
      Y(SI) = F06BLF(V(M), A(M,SI), FAIL)
      IF (FAIL) THEN
         IERR = 3
         RETURN
      END IF
      DO 20 K = M - 1, 1, -1
         SUM = V(K)
         DO 10 J = K + 1, M
            SI = S(J)
            SUM = SUM - A(K,SI) * Y(SI)
   10    CONTINUE
         SI = S(K)
         Y(SI) = F06BLF(SUM, A(K,SI), FAIL)
         IF (FAIL) THEN
            IERR = 3
            RETURN
         END IF
   20 CONTINUE
```

```
      RETURN
C *** Last line of COMPYI ***
      END
```

HHTRAN

```
      SUBROUTINE HHTRAN(MA, NA, L, K, A, LDA, Q, RWORK, TOL)
C
C     PURPOSE
C
C     To compute the Householder reflection Q which transforms a
C     of A into the first unit vector and to apply Q left and right to
C     REMARK: This auxiliary routine is intended to be called only from the
C             routine COLRED.
C
C     ARGUMENTS IN
C
C        MA - INTEGER.
C            The number of rows of matrix A.
C            MA >= 1.
C        NA - INTEGER.
C            The number of columns of matrix A.
C            NA >= MA.
C        L - INTEGER.
C            The index of the column of A to be transformed.
C            L >= 1.
C        K - INTEGER.
C            The row index from which the column is to be transformed.
C            K >= 1.
C        A - DOUBLE PRECISION array of DIMENSION (LDA,NA).
C            The leading MA by NA part of this array must contain the matrix A.
C            The left lower MA - K + 1 by L - 1 block of the matrix A is
C            understood to be zero made by former transformations.
C            Note: this array is overwritten.
C        LDA - INTEGER.
C            The leading dimension of array A as declared in the calling
C            program.
C            LDA >= MA.
C
C     ARGUMENTS OUT
C
C        A - DOUBLE PRECISION array of DIMENSION (LDA,NA).
C            The leading MA by NA part of this array contains the transformed
C            matrix Q A Q , where Q  is the Householder transformation
C                    1   2        i
C            appropriately augmented with an identity matrix.
```

```
C        Q - DOUBLE PRECISION array of DIMENSION at least (MA-K+1).
C            The leading MA - K + 1 elements of this array contain the vector
C            u which defines the Householder reflection (see F06FSF, i.e.,
C            Q(1) contains the value zeta and Q(i), i=2,...MA-K+1 contain the
C            vector z).
C
C     WORKSPACE
C
C        RWORK - DOUBLE PRECISION array of DIMENSION at least (MA-K+1).
C
C     TOLERANCES
C
C        TOL - DOUBLE PRECISION.
C            A tolerance below which matrix elements are considered to be
C            zero.
C
C     METHOD
C
C     Let w be the subcolumn A(i,L), i=K,...,MA. Then the Householder
C     reflection P = I - mu * u * u', where mu = 1/u(1), such that Pw = e1
C     (the first unit vector) is computed.
C     Let
C         Q  = ( I  0 )       Q  = ( I  0 )
C          1   ( 0  P )        2   ( 0  P )
C     be such that Q  is MA by MA and Q  is NA by NA.
C                   1                  2
C     Then Q  A Q   is computed and stored in A.
C           1    2
C
C     CONTRIBUTOR
C
C        A.J. Geurts.
C
C     REVISIONS
C
C        1992, October 27.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER (ZERO = 0.0D0, ONE = 1.0D0)
C     .. Scalar Arguments ..
      INTEGER MA, NA, K, L, LDA
      DOUBLE PRECISION TOL
C     .. Array Arguments ..
      DOUBLE PRECISION A(LDA,NA), Q(MA-K+1), RWORK(MA-K+1)
C     .. Local Scalars ..
      INTEGER LEN, NMK, NL1
```

```
      DOUBLE PRECISION ZETA, MU
C     .. External Subroutines/Functions ..
      EXTERNAL DCOPY, DGER, DGEMV, F06FBF, F06FSF
C
C     .. Executable Statements ..
C
      LEN = MA - K + 1
      IF (LEN .GT. 1) THEN
C
C        Generate the Householder transformation.
C
         CALL DCOPY(LEN, A(K,L), 1, Q, 1)
         CALL F06FSF(LEN-1, Q(1), Q(2), 1, TOL, ZETA)
C
         IF (ZETA .NE. ZERO) THEN
C
C           Compute the matrix Q A by w = A'u and next A = A - mu * uw'.
C                               1
            Q(1) = ZETA
            NL1 = NA - L + 1
            CALL F06FBF(MA, ZERO, RWORK, 1)
            MU = ONE/ZETA
            CALL DGEMV('T', LEN, NL1, ONE, A(K,L), LDA, Q, 1, ZERO,
     *                  RWORK, 1)
            CALL DGER(LEN, NL1, -MU, Q, 1, RWORK, 1, A(K,L), LDA)
C
C           Compute the matrix Q A Q  by w = Au and next A = A - mu * wu'.
C                               1   2
            NMK = NA - MA + K
            CALL DGEMV('N', MA, LEN, ONE, A(1,NMK), LDA, Q, 1, ZERO,
     *                  RWORK, 1)
            CALL DGER(MA, LEN, -MU, RWORK, 1, Q, 1, A(1,NMK), LDA)
         END IF
      ELSE
         Q(1) = ZERO
      END IF
      RETURN
C *** Last line of HHTRAN ***
      END
```

MKPENC

```
      SUBROUTINE MKPENC(MP, NP, DP, P, LDP1, LDP2, A, LDA, MA, NA)
C
C     PURPOSE
C
```

```
C     Given an MP x NP polynomial matrix of degree dp
C                                         dp-1            dp
C     P(s) = P(0) + ... + P(dp-1) * s     + P(dp) * s            (1)
C
C     the subroutine MKPENC constructs the first degree part A of the
C     linearization of the polynomial matrix P(s), where
C
C            | P(dp)     0  .           .   0 |
C            | P(dp-1)   I  0           .   0 |
C        A = |   .       0  I  .        .   0 |                  (2)
C            |   .             .  .  .  .   . |
C            |   .                I  0   0 |
C            | P(0)      0           I   0 |
C
C     REMARK: This auxiliary routine is intended to be called only from the
C             routine COLRED.
C
C     ARGUMENTS IN
C
C        MP - INTEGER.
C            The number of rows of the polynomial matrix P(s).
C            MP >= 1.
C        NP - INTEGER.
C            The number of columns of the polynomial matrix P(s).
C            NP >= 1.
C        DP - INTEGER.
C            The degree of the polynomial matrix P(s).
C            DP >= 1.
C        P - DOUBLE PRECISION array of DIMENSION (LDP1,LP2,DP+1).
C            The leading MP by NP by (DP+1) part of this array must contain
C            the coefficients of the polynomial matrix P(s).  Specifically,
C            P(i,j,k) must contain the coefficient of s**(k-1) of the
C            polynomial which is the (i,j)-th element of P(s), where
C            i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DP+1.
C        LDP1 - INTEGER.
C            The leading dimension of array P as declared in the calling
C            program.
C            LDP1 >= MP.
C        LDP2 - INTEGER.
C            The second dimension of array P as declared in the calling
C            program.
C            LDP2 >= NP.
C
C     ARGUMENTS OUT
C
C        A - DOUBLE PRECISION array of DIMENSION (LDA,(DP+1)*MP+NP).
C            The leading (DP+1)*MP by (DP+1)*MP+NP part of this array
C            contains the matrix A as described in (2).
```

```
C        LDA - INTEGER.
C            The leading dimension of array A as declared in the calling
C            program.
C            LDA >= (DP+1)*MP.
C        MA - INTEGER.
C            The number of rows of matrix A.
C        NA - INTEGER.
C            The number of columns of matrix A.
C
C     CONTRIBUTOR
C
C        A.J.Geurts.
C
C     REVISIONS
C
C        1992, October 27.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO, ONE
      PARAMETER (ZERO = 0.0D0, ONE = 1.0D0)
C     .. Scalar Arguments ..
      INTEGER MP, NP, DP, LDP1, LDP2, LDA, MA, NA
C     .. Array Arguments ..
      DOUBLE PRECISION P(LDP1,LDP2,*), A(LDA,*)
C     .. Local Scalars ..
      INTEGER J, M1, NJ
C     .. External Subroutines/Functions ..
      EXTERNAL F06QFF, F06QHF
C
C     .. Executable Statements ..
C
C     Initialization of the matrix A.
C
      M1 = DP * MP
      MA = M1 + MP
      NA = MA + NP
      CALL F06QHF('G', MP, MA, ZERO, ZERO, A(1,NP+1), LDA)
      CALL F06QHF('G', M1, MA, ZERO, ONE, A(MP+1,NP+1), LDA)
C
C     Insert the matrices P(0), P(1), ..., P(pd) at the right places in A.
C
      NJ = M1 + 1
      DO 20 J = 1, DP + 1
         CALL F06QFF('G', MP, NP, P(1,1,J), LDP1, A(NJ,1), LDA)
         NJ = NJ - MP 20 CONTINUE
C
      RETURN
```

```
C *** Last line of MKPENC ***
      END
```

MULTPM

```
      SUBROUTINE MULTPM(ALPHA, RP1, CP1, CP2, DP1, DP2, DP3, P1, LDP11,
     *                  LDP12, P2, LDP21, LDP22, P3, LDP31, LDP32,
     *                  RWORK, IERR)
C
C     PURPOSE
C
C     To compute the coefficients of the real polynomial matrix
C
C        P(s) = P1(s) * P2(s) + alpha * P3(s),                   (1)
C
C     where P1(s), P2(s) and P3(s) are real polynomial matrices and alpha is
C     a real scalar.
C     Each of the polynomial matrices may by the zero matrix.
C
C     ARGUMENTS IN
C
C        ALPHA - DOUBLE PRECISION.
C           The value of the scalar factor alpha of the problem.
C        RP1 - INTEGER.
C           The number of rows of the matrices P1(s) and P3(s).
C           RP1 >= 1.
C        CP1 - INTEGER.
C           The number of columns of matrix P1(s) and the number of rows
C           of matrix P2(s).
C           CP1 >= 1.
C        CP2 - INTEGER.
C           The number of columns of the matrices P2(s) and P3(s).
C           CP2 >= 1.
C        DP1 - INTEGER.
C           The degree of the polynomial matrix P1(s).
C           DP1 >= -1.
C        DP2 - INTEGER.
C           The degree of the polynomial matrix P2(s).
C           DP2 >= -1.
C        DP3 - INTEGER.
C           The degree of the polynomial matrix P3(s).
C           DP3 >= -1.
C           Note: DP3 is overwritten.
C        P1 - DOUBLE PRECISION array of (LDP11,LDP12,*).
C           If DP1 >= 0, then the leading RP1 by CP1 by (DP1+1) part of this
C           array must contain the coefficients of the polynomial matrix
```

```
C           P1(s). Specifically, P1(i,j,k) must contain the coefficient of
C           s**(k-1) of the polynomial which is the (i,j)-th element of P1(s),
C           where i = 1,2, ...,RP1, j = 1,2,...,CP1 and k = 1,2,...,DP1+1.
C           If DP1 = -1, then P1(s) is taken to be the zero polynomial matrix
C           and the array P1 is not referenced.
C        LDP11 - INTEGER.
C           The leading dimension of array P1 as declared in the calling
C           program.
C           LDP11 >= RP1 if DP1 >= 0.
C           LDP11 >= 1 if DP1 = -1.
C        LDP12 - INTEGER.
C           The second dimension of array P1 as declared in the calling
C           program.
C           LDP12 >= CP1 if DP1 >= 0,
C           LDP12 >= 1 if DP1 = -1.
C        P2 - DOUBLE PRECISION array of (LDP21,LDP22,*).
C           If DP2 >= 0, then the leading CP1 by CP2 by (DP2+1) part of this
C           array must contain the coefficients of the polynomial matrix
C           P2(s). Specifically, P2(i,j,k) must contain the coefficient of
C           s**(k-1) of the polynomial which is the (i,j)-th element of P2(s),
C           where i = 1,2, ...,CP1, j = 1,2,...,CP2 and k = 1,2,...,DP2+1.
C           If DP2 = -1, then P2(s) is taken to be the zero polynomial matrix
C           and the array P2 is not referenced.
C        LDP21 - INTEGER.
C           The leading dimension of array P2 as declared in the calling
C           program.
C           LDP21 >= CP1 if DP2 >= 0.
C           LDP21 >= 1 if DP2 = -1.
C        LDP22 - INTEGER.
C           The second dimension of array P2 as declared in the calling
C           program.
C           LDP22 >= CP2 if DP2 >= 0,
C           LDP22 >= 1 if DP2 = -1.
C        P3 - DOUBLE PRECISION array of (LDP31,LDP32,lenp3),
C           where lenp3 = MAX(DP1+DP2,DP3,0) + 1.
C           If DP3 >= 0, then the leading RP1 by CP2 by (DP3+1) part of this
C           array must contain the coefficients of the polynomial matrix
C           P3(s). Specifically, P3(i,j,k) must contain the coefficient of
C           s**(k-1) of the polynomial which is the (i,j)-th element of P3(s),
C           where i = 1,2, ...,RP1, j = 1,2,...,CP2 and k = 1,2,...,DP3+1.
C           If DP3 = -1, then P3(s) is taken to be the zero polynomial matrix.
C           Note: this array is overwritten.
C        LDP31 - INTEGER.
C           The leading dimension of array P3 as declared in the calling
C           program.
C           LDP31 >= RP1.
C        LDP32 - INTEGER.
C           The second dimension of array P3 as declared in the calling
```

```
C           program.
C           LDP32 >= CP2.
C
C     ARGUMENTS OUT
C
C        DP3 - INTEGER.
C           The degree of the resulting polynomial matrix P(s).
C        P3 - DOUBLE PRECISION array of DIMENSION (LDP31,LDP32,lenp3).
C           If DP3 >= 0 on exit, then the leading RP1 by CP2 by (DP3+1) part
C           of this array contains the coefficients of P(s).  Specifically,
C           P3(i,j,k) contains the coefficient of s**(k-1) of the polynomial
C           which is the (i,j)-th element of P(s), where i = 1,2, ...,RP1,
C           j = 1,2,...,CP2 and k = 1,2,...,DP3+1.
C           If DP3 = -1 on exit, then P(s) is the zero polynomial matrix and
C           the contents of the array P3 are undefined.
C
C     WORK SPACE
C
C        RWORK - DOUBLE PRECISION array of DIMENSION at least (CP1).
C
C     ERROR INDICATOR
C
C        IERR - INTEGER.
C           Unless the routine detects an error (see next section),
C           IERR contains 0 on exit.
C
C     WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C        IERR = 1 : Invalid input parameter(s).
C
C     METHOD
C
C     Given the real polynomial matrices
C                DP1            i           DP2            i
C        P1(s) = SUM (a(i+1) * s ), P2(s) = SUM (b(i+1) * s ),
C                i=0                        i=0
C                DP3            i
C        P3(s) = SUM (c(i+1) * s ).
C                i=0
C     and a real scalar alpha, the routine computes the coefficients
C     d(1), d(2), ... of the polynomial matrix (1) from the formula
C                   s
C        d(i+1) := SUM (a(k+1) * b(i-k+1)) + alpha * c(i+1),
C                  k=r
C     where i = 0,1,...,DP1+DP2 and r and s depend on the value of i,
C     i.e. for r <= k <= s both a(k+1) and b(i-k+1) must exist.
C
C     CONTRIBUTOR
```

```
C
C        A.J. Geurts (Eindhoven University of Technology).
C
C     REVISIONS
C
C        1992, October 28.
C
      IMPLICIT NONE
C     .. Parameters ..
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO = 0.0D0)
C     .. Scalar Arguments ..
      INTEGER RP1, CP1, CP2, DP1, DP2, DP3, LDP11, LDP12,
     *        LDP21, LDP22, LDP31, LDP32, IERR
      DOUBLE PRECISION ALPHA
C     .. Array Arguments ..
      DOUBLE PRECISION P1(LDP11, LDP12, *), P2(LDP21, LDP22, *),
     *                 P3(LDP31, LDP32, *), RWORK(*)
C     .. Local Scalars ..
      INTEGER H, I, J, K, DPOL3, E
      DOUBLE PRECISION W
      LOGICAL CFZERO
C     .. External Subroutines/Functions ..
      EXTERNAL DCOPY, DDOT, DSCAL, F06FBF
      DOUBLE PRECISION DDOT
C
C     .. Executable Statements ..
C
C     Check input parameters.
C
      IF ((RP1.LT.1) .OR. (CP1.LT.1) .OR. (CP2.LT.1)
     *   .OR. (DP1.LT.-1) .OR. (DP2.LT.-1) .OR. (DP3.LT.-1)
     *   .OR. ((LDP11.LT.RP1) .AND. (DP1.GE.0))
     *   .OR. ((LDP11.LT.1) .AND. (DP1.EQ.-1))
     *   .OR. ((LDP12.LT.CP1) .AND. (DP1.GE.0))
     *   .OR. ((LDP12.LT.1) .AND. (DP1.EQ.-1))
     *   .OR. ((LDP21.LT.CP1) .AND. (DP2.GE.0))
     *   .OR. ((LDP21.LT.1) .AND. (DP2.EQ.-1))
     *   .OR. ((LDP22.LT.CP2) .AND. (DP2.GE.0))
     *   .OR. ((LDP22.LT.1) .AND. (DP2.EQ.-1))
     *   .OR. (LDP31.LT.RP1) .OR. (LDP32.LT.CP2)) THEN
         IERR = 1
         RETURN
      END IF
C
      IERR = 0
      IF (ALPHA .EQ. ZERO) THEN
         DP3 = -1
```

```
      END IF
C
      IF (DP3 .GE. 0) THEN
C
C        P3(s) := ALPHA * P3(s).
C
         DO 20 K = 1, DP3 + 1
            DO 10 J = 1, CP2
               CALL DSCAL(RP1, ALPHA, P3(1,J,K), 1)
   10       CONTINUE
   20    CONTINUE
      END IF
C
      IF ((DP1 .EQ. -1) .OR. (DP2 .EQ. -1)) RETURN
C
C     Neither of P1(s) and P2(s) is the zero polynomial.
C
      DPOL3 = DP1 + DP2
      IF (DPOL3 .GT. DP3) THEN
C
C        Initialize the additional part of P3(s) to zero.
C
         DO 40 K = DP3 + 2, DPOL3 + 1
            DO 30 J = 1, CP2
               CALL F06FBF(RP1, ZERO, P3(1,J,K), 1)
   30       CONTINUE
   40    CONTINUE
         DP3 = DPOL3
      END IF
C                                                            k-1
C     The inner product of the j-th row of the coefficient of s    of P1(s)
C                                                  i-1
C     and the h-th column of the coefficient of s     of P2(s) contributes to
C                                                   k+i-2
C     the (j,h)-th element of the coefficient of s       of P3(s).
C
      DO 80 K = 1, DP1 + 1
         DO 70 J = 1, RP1
            CALL DCOPY(CP1, P1(J,1,K), LDP11, RWORK, 1)
            DO 60 I = 1, DP2 + 1
               E = K + I - 1
               DO 50 H = 1, CP2
                  W = DDOT(CP1, RWORK, 1, P2(1,H,I), 1)
                  P3(J,H,E) = W + P3(J,H,E)
   50          CONTINUE
   60       CONTINUE
   70    CONTINUE
   80 CONTINUE
```

```
C
C     Computation of the exact degree of P3(s).
C
      CFZERO = .TRUE.
C     WHILE (DP3 >= 0 and CFZERO) DO
   90 IF ((DP3 .GE. 0) .AND. CFZERO) THEN
         DPOL3 = DP3 + 1
         DO 110 I = 1, RP1
            DO 100 J = 1, CP2
               IF (P3(I,J,DPOL3) .NE. ZERO) CFZERO = .FALSE.
  100       CONTINUE
  110    CONTINUE
         IF (CFZERO) DP3 = DP3 - 1
         GO TO 90
      END IF
C     END WHILE 90
C
      RETURN
C *** Last line of MULTPM ***
      END
```

PRMAPO

```
      SUBROUTINE PRMAPO(MP, NP, DP, L, NOUT, P, LDP1, LDP2, TEXT, IERR)
C
C     PURPOSE
C
C     To print the MP by NP coefficient matrices of a matrix polynomial
C                                                         dp-1          dp
C        P(s) = P(0) + P(1) * s + . . . P(dp-1) * s    + P(dp) * s   .
C
C     The elements of the matrices are output to 7 significant figures.
C
C     ARGUMENTS IN
C
C        MP - INTEGER.
C            The number of rows of the matrix polynomial P(s).
C            MP >= 1.
C        NP - INTEGER.
C            The number of columns of the matrix polynomial P(s).
C            NP >= 1.
C        DP - INTEGER.
C            The degree of the matrix polynomial P(s).
C            DP >= 0.
C        L - INTEGER.
C            The number of elements of the coefficient matrices to be
```

```
C             printed per line.
C             1 <= L <= 5.
C         NOUT - INTEGER.
C             The output channel to which the results are sent.
C             NOUT >= 0.
C         P - DOUBLE PRECISION array of DIMENSION (LDP1,LDP2,DP+1).
C             The leading MP by NP by (DP+1) part of this array must contain
C             the coefficients of the matrix polynomial P(s).  Specifically,
C             P(i,j,k) must contain the coefficient of s**(k-1) of the
C             polynomial which is the (i,j)-th element of P(s), where
C             i = 1,2,...,MP, j = 1,2,...,NP and k = 1,2,...,DP+1.
C         LDP1 - INTEGER.
C             The leading dimension of array P as declared in the calling
C             program.
C             LDP1 >= MP.
C         LDP2 - INTEGER.
C             The second dimension of array P as declared in the calling
C             program.
C             LDP2 >= NP.
C         TEXT - CHARACTER*72.
C             Title caption of the coefficient matrices to be printed.
C             TEXT is followed by the degree of the coefficient matrix,
C             within brackets. If TEXT = ' ', then the coefficient matrices
C             are separated by an empty line.
C
C      ERROR INDICATOR
C
C         IERR - INTEGER.
C             Unless the routine detects an error (see next section),
C             IERR contains 0 on exit.
C
C      WARNINGS AND ERRORS DETECTED BY THE ROUTINE
C
C         IERR = 1 : Invalid input parameter(s).
C
C      METHOD
C
C      For i = 1, 2, ..., DP + 1 the routine first prints the contents of
C      TEXT followed by (i-1) as a title, followed by the elements of the
C      MP by NP coefficient matrix P(i) such that
C      (i)  if NP < L, then the leading MP by NP part is printed;
C      (ii) if NP = k*L + p (where k, p > 0), then k MP by L blocks of
C           consecutive columns of P(i) are printed one after another
C           followed by one MP by p block containing the last p columns of P(i).
C      Row numbers are printed on the left of each row and a column number on
C      top of each column.
C
C      CONTRIBUTOR
```

```
C
C        A.J. Geurts (Eindhoven University of Technology).
C
C     REVISIONS
C
C        1992, October 28.
C
      IMPLICIT NONE
C     .. Scalar Arguments ..
      INTEGER MP, NP, DP, L, NOUT, LDP1, LDP2, IERR
      CHARACTER*(*) TEXT
C     .. Array Arguments ..
      DOUBLE PRECISION P(LDP1,LDP2,*)
C     .. Local Scalars ..
      INTEGER I, J, J1, J2, JJ, K, LENTXT, LTEXT, N1
C     .. Intrinsic Functions ..
      INTRINSIC LEN, MIN
C
C     .. Executable Statements ..
C
C     Check input parameters.
C
      LENTXT = LEN(TEXT)
      LTEXT = MIN(72,LENTXT)
C     WHILE (TEXT(LTEXT:LTEXT) =  ' ') DO
   10 IF (TEXT(LTEXT:LTEXT) .EQ. ' ') THEN
         LTEXT = LTEXT - 1
         GO TO 10
      END IF
C     END WHILE 10
C
      IF (MP.LT.1 .OR. NP.LT.1 .OR. DP.LT.0 .OR. L.LT.1 .OR. L.GT.5
     *    .OR. NOUT.LT.0 .OR. LDP1.LT.MP .OR. LDP2.LT.NP) THEN
         IERR = 1
         RETURN
      END IF
C
      IERR = 0
C
      DO 50 K = 1, DP + 1
         IF (LTEXT .EQ. 0) THEN
            WRITE (NOUT, FMT = 99999)
         ELSE
            WRITE (NOUT, FMT = 99998) TEXT(1:LTEXT), K - 1, MP, NP
         END IF
         N1 = (NP - 1)/L
         J1 = 1
         J2 = L
```

```
         DO 30 J = 1, N1
            WRITE (NOUT, FMT = 99997) (JJ, JJ = J1, J2)
            DO 20 I = 1, MP
               WRITE (NOUT, FMT = 99996) I, (P(I,JJ,K), JJ = J1, J2)
   20       CONTINUE
            J1 = J1 + L
            J2 = J2 + L
   30    CONTINUE
         WRITE (NOUT, FMT = 99997) (J, J = J1, NP)
         DO 40 I = 1, MP
            WRITE (NOUT, FMT = 99996) I, (P(I,JJ,K), JJ = J1, NP)
   40    CONTINUE
   50 CONTINUE
      WRITE (NOUT, FMT = 99999)
C
      RETURN
99999 FORMAT (' ')
99998 FORMAT (/, 1X, A, '(', I2, ')', ' (', I2, 'X', I2, ')')
99997 FORMAT (5X, 5(6X, I2, 7X))
99996 FORMAT (1X, I2, 2X, 5D15.7)
C *** Last line of PRMAPO ***
      END
```