Polynomial Control Systems

MEHMET TURAN SÖYLEMEZ and İLKER ÜSTOĞLU

Polynomial Control Systems (PCS), a Mathematica-based toolbox released in July 2006 by Wolfram Research, Inc., expands the functionality of Wolfram's Control System Professional Suite (CSPS) in a variety of ways. CSPS, which now consists of Control System Professional (CSP), Advanced Numerical Methods (ANM), and the new Polynomial Control Systems, was first released in the middle of 2003 [1] and is used by professionals and academicians working on control systems. PCS presents new tools for the modeling, analysis, and design of linear control systems described by polynomial matrix equations or matrices with rational polynomial entries [2]. Like its counterparts CSP and ANM, PCS follows an object-oriented methodology for analyzing and designing control systems.

In this product review, some of the new tools provided by PCS are examined using tutorial examples and compared to similar tools available in Matlab. Specifically, the Control Systems Toolbox (version 6.1) [3] and Polynomial Control System Toolbox (PolyX—version 2.5) [4] of Matlab are used in comparisons. However, we point out that this review should not be considered as a comprehensive comparative study of PCS since our focus is to introduce some capabilities of PCS and, when feasible, contrast comparable capabilities of Matlab with PCS.

Polynomial-based algorithms can provide useful insights for the analysis and design of multivariable systems. In the first part of the review, we examine several new system descriptions supported by PCS and the corresponding analysis tools. In the second part of the review, we focus on the design and synthesis of multivariable systems using PCS.

MODELING AND ANALYSIS OF LINEAR SYSTEMS

Like CSP, PCS deals with state-space and transfer-function models of both continuous- and discrete-time systems. Furthermore, PCS provides new models such as *SystemMatrix*, *LeftMatrixFraction*, and *RightMatrixFraction*. All of these model objects can be converted from one form to another. Recall that the Laplace transform of a system of ordinary differential equations that describe the behavior of a linear time-invariant (LTI) system can be written as

$$\begin{aligned} T(s)\xi &= U(s)u, \\ y &= V(s)\xi + W(s)u, \end{aligned} \tag{1}$$

where s is the Laplace variable and ξ, u, and y are vectors of Laplace-transformed state variables, inputs, and outputs, respectively. The terms $T(s), U(s), V(s)$, and $W(s)$ in (1) are polynomial matrices of suitable dimensions that define the system behavior. The system matrix corresponding to (2) is defined as

$$P(s) = \begin{bmatrix} T(s) & U(s) \\ -V(s) & W(s) \end{bmatrix}. \tag{2}$$

It is important to note that the dimension of the matrix $T(s)$ must be adjusted so that it is greater than or equal to the degree of its determinant. The system matrix representation given in (2) can be thought as a generalization of the usual state-space representation wherein $T(s)$ takes the special form $T(s) = sI - A$, and $U(s), V(s)$, and $W(s)$ are constant matrices independent of the Laplace variable s. As a result, using the system matrix representation of (2), it is possible to examine internal states and associated characteristics of a system [5].

To illustrate the modeling and analysis capabilities of PCS, we begin by loading the PCS toolbox. In addition, we load Mathematica's *MatrixManipulation* toolbox, which is required for using some functions of PCS.

```
In[1]:= << PolynomialControlSystems'
        << LinearAlgebra 'MatrixManipulation'
```

Next, we set up a two-input, two-output system of the form (1).

```
In[3]:= T = ((s + 1) (s + 2) (s + 3) (s + 4)    2(s + 1) (s + 2) (s + 4)
             (s + 1) (s + 3) (s + 4)            (s + 1) (s + 4) (s + 5));
        U = (s + 2  s + 2
             1      s + 4);
        V = ((s + 3) (s + 4)    2(s + 4)
             5(s + 3)           2(s + 4));
        W = ZeroMatrix [2, 2] ;
```

The corresponding system matrix $P(s)$ is obtained as follows:

1066-033X/07/$25.00©2007IEEE

In[7]:=
```
P = SystemMatrix[s, T, U, V, W]
```

Out[7]=

$$\left(\begin{array}{ccccccc|cc} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&(s+1)(s+2)(s+3)(s+4)&2(s+1)(s+2)(s+4)&s+2&s+2\\ 0&0&0&0&0&(s+1)(s+3)(s+4)&(s+1)(s+4)(s+5)&0&s+4\\ \hline 0&0&0&0&0&-(s+3)(s+4)&-2(s+4)&0&0\\ 0&0&0&0&0&-5(s+3)&-2(s+4)&0&0 \end{array}\right)^{\mathcal{M}}$$

Note that the upper case $\mathcal{M}$ at the upper right corner of the matrix indicates that it is a system matrix object. Note also that, since $\det(T(s))$, where $\det(\cdot)$ is the determinant operator, is a seventh-order polynomial and $T(s)$ is 2×2, the SystemMatrix function introduces five dummy variables to adjust the sizes of $T(s)$, $U(s)$, and $V(s)$. The internal modes of the system are determined from the roots of $\det(T(s)) = 0$ as shown below.

In[8]:=
```
s / . Solve[Det [T] == 0, s ] (* Internal modes *)
```

Out[8]= {–4, –4, –3, –3, –2, –1, –1}

It is possible to obtain the Smith form of a polynomial matrix $P(s)$ as follows:

In[9]:=
```
SF = SmithForm [P]
```

Out[9]=

$$\begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&s+3&0\\ 0&0&0&0&0&0&0&(s-1)(s+2)(s+3)(s+4) \end{pmatrix}$$

The polynomials on the diagonal of the Smith form of the system matrix are the *invariant polynomials* of the system. The roots of the invariant polynomials give the *invariant zeros*, that is, zeros that do not change under similarity transformations or feedback. Invariant zeros of the system matrix object can also be computed directly from a system matrix description as shown below:

In[1]:=
```
InvariantZeros [P]
```

Out[10]= {–4, –3, –3, –2, 1}

As with almost all commands of Mathematica, the underlying calculations are kept exact when the input arguments to PCS commands are exact (integer, rational, or symbolic expressions). For example, it is possible to obtain symbolic results for the Smith form or invariant zeros of a system in terms of symbolic parameters. Since some algorithms, such as the algorithm that calculates the Smith form, can be sensitive to numerical errors, the ability to use exact calculations is an advantage of PCS. If we use a numerical version of the same system matrix, as shown below, machine precision calculations result in a rather different Smith form.

In[11]:=
```
SmithForm [P // N]
```

Out[11]=

$$\begin{pmatrix} 1.&0.&0.&0.&0.&0.&0.&0.&0.\\ 0.&1.&0.&0.&0.&0.&0.&0.&0.\\ 0.&0.&1.&0.&0.&0.&0.&0.&0.\\ 0.&0.&0.&1.&0.&0.&0.&0.&0.\\ 0.&0.&0.&0.&1.&0.&0.&0.&0.\\ 0.&0.&0.&0.&0.&1.&0.&0.&0.\\ 0.&0.&0.&0.&0.&0.&1.&0.&0.\\ 0.&0.&0.&0.&0.&0.&0.&1.&0.\\ 0.&0.&0.&0.&0.&0.&0.&0.&1.\,(s-1.)(s+2.)(s+3.)^2(s+4.) \end{pmatrix}$$

To the best of our knowledge there is no standard Matlab toolbox that supports the system matrix representation given in (2). Using PolyX [4], however, it is possible to obtain similar results. Specifically, using the following procedure the polynomial matrix representation of (2) can be produced in Matlab.

```
>> T=[24+50*s+35*s^2+10*s^3+s^4,16+28*s+14*s^2+2*s^3
    12+19*s+8*s^2+s^3,20+29*s+10*s^2+s^3];
U=[2+s,2+s
   1,4+s];
V=[12+7*s+s^2,8+2*s
  15+5*s,8+2*s];
W=zeros(2,2);
Te=[eye(5) zeros(5,2); zeros(2,5) T];  %T, U, V  with
Ue=[zeros(5,2);U];                     %dummy variables
Ve=[zeros(2,5) V];                     %introduced
P=[Te,Ue;–Ve,W];  %system matrix in polynomial form
```

Next, it is possible to calculate the internal modes of the system as follows:

```
>> intmod=roots (det (T))   %internal modes

intmod =

   –4.0000
   –4.0000
   –3.0000
   –3.0000
   –2.0000
   –1.0000
   –1.0000
```

The corresponding Smith form is shown below:

```
>> SP=smith(P)          %smith form of the system matrix
Warning: The relative residue of calculation is  23.0583
> In  smith at 300


SP =

   Columns 1 through 8
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     1
     0     0     0     0     0     0     0     0

   Column 9
     0
     0
     0
     0
     0
     0
     0
     0
     -72 – 30s + 49s^2 + 41s ^3 + 11s^4 + s^5
```

Note that the same result is produced by PCS with a numerical input. Unlike PCS, PolyX produces a warning message to indicate a possible problem in calculations. There is, however, no difference in the invariant zeros, which are found from the roots of the determinant of the system matrix $P(s)$ or its Smith form.

```
>> invz = roots (P)  %invariant zeros

invz =
      1.0000
     –4.0000
     –3.0000  + 0.00001
     –3.0000  – 0.00001
     –2.0000
```

Returning to PCS, by using the function *LeftCoprime[]*, it is possible to check whether $T(s)$ and $U(s)$ are coprime.

In[12]:= LeftCoprime [T, U, s]

Out[12]= False

Since $T(s)$ and $U(s)$ are not left coprime they can be expressed as

$$T(s) = L(s)T_1(s),$$
$$U(s) = L(s)U_1(s),$$

where $L(s)$ is the left matrix greatest common divisor. For this purpose the function *LeftGCDDecomposition[]* is used.

In[13]:= {L, T_1, U_1} = LeftGCDDecomposition [T, U, s];

Out[14]:= $L = \begin{pmatrix} s+2 & 0 \\ -s-2 & s+3 \end{pmatrix}$

The roots of the determinant of $L(s)$, which are the *input decoupling zeros* of the system, form a subset of the invariant zeros and correspond to the uncontrollable modes of a system arising from a state-space model. The input decoupling zeros of the system given above are -2 and -3. By applying the function *InputDecouplingZeros[]* to the system matrix object, the same result can be obtained.

In[15]:= InputDecouplingZeros [P]

Out[15]= {–3, –2}

It is possible to obtain the same result in Matlab as follows:

```
>> L=gld (Te,Ue);   %left matrix GCD for T and U
idz=roots (L)   %input decoupling zeros

idz =

     –3.0000
     –2.0000
```

Similar commands are available in PCS for finding the right greatest common divisor and the output decoupling zeros, which correspond to unobservable modes of a state-space system. See the following example.

In[16]:= OutputDecouplingZeros [P]

Out[16]= {–4, –3}

The system matrix with input decoupling zeros removed

$$P_1(s) = \begin{bmatrix} T_1(s) & U_1(s) \\ -V(s) & W(s) \end{bmatrix},$$

can be constructed by the command P_1=*SystemMatrix* $[s, T_1, U_1, V_1, W]$. The same result can be achieved by using the *RemoveInputDecouplingZeros[P]* function. The least-order system can be obtained by successively removing input and output decoupling zeros.

In[17]:=

```
P_L = LeastOrderSystem [P]
```

Out[17]=

$$\left(\begin{array}{ccccc|cc|cc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & s^2+5s+4 & 2(s+1) & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & s^2+5s+4 & 3(s+1) & 1 & 2 \\ \hline 0 & 0 & 0 & 0 & 0 & -s-4 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -5 & -2 & 0 & 0 \end{array}\right)_{\mathcal{M}}$$

Note that the modes of the least-order system are the poles, since there are no hidden modes in this representation.

In[18]:=

```
T_L = P_L [s] [[1, 1]];   (*T for least order sys*)
ples = s / . Solve [Det [T_L] = 0]
```

Out[19]=

```
{–4, –1, –1}
```

The Matlab code for obtaining a least-order system is given as follows:

```
>> T1=L\Te;  %Te=L*T1
U1=L\Ue;  %Ue=L*U1
P1=[T1, U1; –Ve, W];   %System with input decoupling
                       %zeros removed.
R2=grd (T1, Ve);       %right matrix GCD for T1 and –Ve
T2=T1/R2; %T1=T2*R2
V2=Ve/R2; %Ve=V2*R2
PL=[T2, U1; –V2, W]    %least order system

PL =

     1 + s          0          0     0     0     0     0     0     1
     2 + 2s         4 + 5s + s^2  0     0     0     0     0     1     1
     0              0          1     0     0     0     0     0     0
     0              0          0     1     0     0     0     0     0
     0              0          0     0     1     0     0     0     0
     0              0          0     0     0     1     0     0     0
     0              0          0     0     0     0     1     0     0
   – 2            – 4 – s      0     0     0     0     0     0     0
   – 2            – 5          0     0     0     0     0     0     0
```

The poles are obtained similarly as shown below.

```
>> poles=roots (T2)    %poles of the system

poles =

   –4.0000
   –1.0000
   –1.0000
```

Conversion of a system matrix object to another kind of representation, such as transfer function or state space, is straightforward in PCS. For example, the transfer function corresponding to the system matrix description given in (1) can be calculated as

$$G(s) = \frac{y(s)}{u(s)} = V(s)T^{-1}(s)U(s) + W(s).$$

A simple way to form this representation is to apply the *TransferFunction[]* command to the model object.

In[20]:=

```
G = TransferFunction [P]
```

Out[20]=

$$\begin{pmatrix} \dfrac{1}{s+1} & \dfrac{1}{s+1} \\ \dfrac{5}{s^2+5s+4} & \dfrac{2s+3}{s^2+5s+4} \end{pmatrix}^{\mathcal{T}}$$

The Matlab code for finding the same transfer function is as follows:

```
>> [adjT2, denG]=inv (T2);  %adjoint and determinant of T2
numG=V2 *adjT2*U1+W;  %numerator part of the transfer function
G=tf (numG, eye (2) *denG)   %transfer function

Transfer function from input 1 to output...
          1
#1:   -------
      s + 1

              5
#2:   ------------
      s^2 + 5s + 4

Transfer function from input 2 to output...
          1
#1:   -------
      s + 1

          2s + 4
#2:   ------------
      s^2 + 5s + 4
```

We generally prefer to use the least-order system matrix representation in deriving the transfer function in Matlab since the original system results in a higher-order transfer function due to inaccuracies introduced by machine size precision calculations.

A canonical form related with rational matrices is the McMillan form, which provides essential information about the poles and transmission zeros of a system. PCS calculates the McMillan form using the *McMillanForm[]* function.

In[21]:= McMillanForm [G]

Out[21]=

$$\begin{pmatrix} \frac{1}{(s+1)\,(s+4)} & 0 \\ 0 & \frac{s-1}{s+1} \end{pmatrix}$$

It is possible to check that the roots of the denominator polynomials of the diagonal entries of the McMillan form give the poles of the system. The roots of the numerator polynomials of the diagonal entries, on the other hand, give the transmission zeros of the system, which provide vital information on the nature of the system, such as insensitivity to certain input frequencies, or stability and performance expectations from the closed-loop system with high loop gains. Transmission zeros can also be found directly from the original system matrix object $P(s)$ or the transfer function object $G(s)$.

In[22]:= TransmissionZeros [P]

Out[22]= {1}

Although the Matlab toolboxes we consider in this review do not support it directly, the McMillan form can be deduced using the Smith form of the numerator part of the transfer function.

```
>> SG=smith (numG);
MG=tf (SG,eye (2)*denG)  %McMillan form

Transfer function from input 1 to output...
                1
#1:  ------------
     s^2 + 5s + 4

#2:  0

Transfer function from input 2 to output...
#1:  0

#2:  s - 1
     -----
     s + 1
```

Transmission zeros, on the other hand, can be calculated from the roots of the determinant of the least-order system matrix.

```
>> transz=roots (PL)    %transmission zeros
transz =
    1.0000
```

Linear time-invariant systems can also be described using matrix fraction descriptions. A left matrix fraction description is given by two polynomial matrices $N_L(s)$ and $D_L(s)$ such that $G(s) = D_L^{-1}(s)N_L(s)$, where $G(s)$ is the transfer function of the system. Similarly, a right matrix fraction description is formulated as $G(s) = N_R(s)D_R^{-1}(s)$. *LeftMatrixFraction[]* and *RightMatrixFraction[]* commands construct left and right matrix-fraction objects from two polynomial matrices or any of the other system objects. The following command, for instance, constructs a new left matrix fraction object.

In[23]:=

$$D_L = \begin{pmatrix} (s+1) & 0 \\ 0 & (s+1)\,(s+4) \end{pmatrix};$$

$$N_L = \begin{pmatrix} 1 & 1 \\ 5 & 2s+3 \end{pmatrix};$$

LMF = LeftMatrixFraction [s, D_L, N_L]

Out[25]:=

$$\begin{pmatrix} s+1 & 0 \\ 0 & (s+1)(s+4) \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 \\ 5 & 2s+3 \end{pmatrix}\Big|^{\mathcal{L}}$$

The upper case $\mathcal{L}$ on the upper right corner of the object denotes that it is a left matrix fraction. It is possible to convert matrix fraction descriptions to other types of system descriptions and vice versa. For example, transfer function of the system described above can be found as

In[26]:= TransferFunction [LMF]

Out[26]=

$$\begin{pmatrix} \frac{1}{s+1} & \frac{1}{s+1} \\ \frac{5}{s^2+5s+4} & \frac{2s+3}{s^2+5s+4} \end{pmatrix}^{\mathcal{T}}$$

which is the same as the transfer function of the system matrix $P(s)$ given above. A right matrix fraction description for $P(s)$ can be found as shown below.

```
In[27]:= RMF = RightMatrixFraction [P]
Out[27]= ( s + 4   s + 4 ) ( s^2 + 5s + 4      0          )^-1 | R
         (  5      2s + 3 ) (      0          (s + 1)(s + 4) )
```

The matrix fraction descriptions found as a result of these conversions are not necessarily coprime.

```
In[28]:= RightCoprime [RMF]
Out[28]= False
```

PolyX toolbox of Matlab also provides various ways of converting different system descriptions. To obtain a right matrix fraction description of the system, for instance, the following command can be used.

```
>> [NR, DR] = rat2rmf (numG, denG*ones (2, 2))   %RMF description

NR =

        0.44 + 0.11s        0.58
        0.55                0.58

DR =

        0.44 + 0.55s + 0.11s^2        0.29 + 0.29s
        0                             0.29 + 0.29s
```

It is possible to use many of the commands introduced above on matrix fraction descriptions directly. We also note that the new system objects (system matrix and matrix fraction description) are compatible with most of the CSP and ANM commands. It is possible, for example, to obtain the output response or extract the subsystems of a system matrix object directly. We skip the details of such compatibilities due to space considerations, and are contented with just mentioning that PCS is integrated into CSPS in a natural way.

CONTROLLER DESIGN

PCS provides several design and synthesis tools to help the control design process, especially for multi-input, multi-output (MIMO) systems. In the following, some control design functions of PCS are demonstrated through an illustrative example.

State-Feedback Pole Assignment

Given the state-space matrices $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$, where n and m are the number of states and inputs, respectively, the state-feedback pole assignment problem is to find a constant state-feedback matrix $K_s \in \mathbb{R}^{m\times n}$ such that the eigenvalues of the closed-loop system matrix $A_c = A - BK_s$ are at desired locations. To illustrate some of the algorithms of PCS, we consider the state-space description of an aircraft model from [6], which has three inputs, namely, spoiler angle, forward acceleration, and elevator degree, as well as five states, namely, altitude, forward speed, pitch angle, pitch rate, and vertical speed.

```
In[29]:= As = ( 0   0        1.132    0        −1     )
              ( 0   −0.0538   −0.1712   0        0.0705 )
              ( 0   0         0        1        0      )
              ( 0   0.0485    0        −0.8556  −1.013 )
              ( 0   −0.2909   0        1.0532   −0.6859 ) ;

         Bs = (  0      0   0      )
              ( −0.12   1   0      )
              (  0      0   0      )
              (  4.419  0   −1.665 )
              (  1.575  0   −0.0732 ) ; (*input matrix*)

         Cs = ( 1 0 0 0 0 )
              ( 0 1 0 0 0 ) ; (*output matrix*)
              ( 0 0 1 0 0 )

         aircraft = StateSpace [As, Bs, Cs,]

Out[30]= (  0   0        1.132    0        −1      |  0   0   0  )S
         (  0   −0.0538  −0.1712  0        0.0705  | −0.12 1   0  )
         (  0   0         0       1        0       |  0   0   0  )
         (  0   0.0485    0       −0.8556  −1.013  |  4.419 0 −1.665 )
         (  0   −0.2909   0       1.0532   −0.6859 |  1.575 0 −0.0732 )
         ( ------------------------------------------------------ )
         (  0   0         0       0        0       |  0   0   0  )
         (  0   1         0       0        0       |  0   0   0  )
         (  0   0         1       0        0       |  0   0   0  ).
```

Although it is not necessary to provide the state-space output matrix C_s for calculations of the state-feedback matrices, it is given in the above system description since this information is used in later subsections when observer design and output-feedback pole assignment are considered. Note that the first three states of the system are taken as system outputs in this example.

For pole assignment, PCS requires that the system description be given in state-space format. Assuming that all of the states are available for measurement, it is possible to design a constant state-feedback controller K_s that places the closed-loop system poles as required if, and only if, the system is controllable [7].

```
In[31]:= Controllable [aircraft]
Out[31]= True
```

Next, the desired closed-loop system poles are specified as follows:

```
In[32]:= desiredCLSPoles = {−1 −Ί, −1 +Ί, −5, −6, −7};
```

PCS extends the StateFeedbackGains[] command of CSP by introducing new methods, namely, *Mapping*, *Spectral*, and *FullRank*, for calculating the state-feedback matrix that yields the desired closed-loop system poles. The *Mapping* and *Spectral* methods together with the *Ackermann* method of CSP produce dyadic (that is, rank-one) state-feedback matrices that can be decomposed as

$$K_s = f\,k^T,$$

where the fan-out vector $f \in \mathbb{R}^{n\times 1}$ determines the weights of the control inputs, and $k^T \in \mathbb{R}^{1\times n}$ is a row vector, which is usually calculated as the state-feedback compensator for the pseudosingle input system (A, Bf). Such a decomposition of the state-feedback matrix allows a simpler structure in the realization stage and is therefore preferred in some cases. Our experience tells us that the *Spectral* method usually produces more accurate results when numerical calculations are carried out, whereas the *Mapping* method usually gives the unique result faster when the underlying calculations are carried out symbolically [8].

Now we use the *Rationalize[]* command to provide an exact description of the system and hence ensure that the calculations are carried out symbolically.

```
>> A = [0, 0, 1.132, 0, –1; ...
    0, –0.0538, –0.1712, 0, 0.0705; ...
    0, 0, 0, 1, 0; ...
    0, 0.0485, 0, –0.8556, –1.013; ...
    0, –0.2909, 0, 1.0532, –0.6859];
B = [0, 0, 0; ...
    –0.12, 1, 0; ...
    0, 0, 0 ; ...
    4.419, 0, –1.665; ...
    1.575, 0, –0.0732];
C = [1, 0, 0, 0, 0; ...
    0, 1, 0, 0, 0; ...
    0, 0, 1, 0, 0];
desiredCLSPoles = [–1–i, –1+i, –5, –6, –7];
```

```
>> %Dyadic State-feedback compensation
f = [0 ; 1; 0];  %fan-out vector
k1 = place (A, B*f, desiredCLSPoles);
Ks1 = f*k1

Ks1 =
   1.0e + 003 *
         0         0         0         0         0
    1.1314    0.0184    1.0612   –0.0991   –0.4114
         0         0         0         0         0
```

In[32]:=
```
desiredCLSPoles = {–1 –i̇, –1 +i̇, –5, –6, –7};
```

In[33]:=
```
Ks1 = StateFeedbackGains [Rationalize[aircraft], desiredCLSPoles, Method → Mapping,
ControlInput → {0, 1, 0} ]
```

Out[33]=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ \frac{700000000000}{618728277} & \frac{184047}{10000} & \frac{269172376652350260 1477}{2536382602204430625} & -\frac{863045497664136284 95374418230769}{870547973997074591128664812500} & -\frac{716310265199859562414255619330 87}{174109594799414918225732962500} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Here, the fan-out vector is selected as $f = [0\ 1\ 0]^T$ using the *ControlInput* option of *StateFeedbackGains* command. Note that a symbolic fan-out vector such as $f = [1\ k_1\ k_2]$ results in a symbolic result, which can be used to address additional design criteria. It is possible to validate that the matrix K_{s1} expressed in *Out[35]* indeed yields the desired closed-loop system poles.

In[34]:=
```
Ts1 = StateFeedbackConnect [aircraft, Ks1] ;
Eigenvalues [Ts1[ [1] ] ]
```

Out[35]=
```
{–7., –6., –5., –1. +1. i, –1. –1. i}
```

A similar result is achieved through the Matlab Control Systems Toolbox using the following commands.

The resulting compensator is numerically the same as the one found by PCS. When the order of the system is small (say, less than ten), there is usually no significant difference between numerical and exact calculations of PCS. However, for higher order systems, numerical results can deviate from the exact results significantly [9], while exact calculations are considerably slower.

The *FullRank* method of *StateFeedbackGains[]* command of PCS uses the Luenberger controllable canonical form (*LuenbergerControllableForm[]*) to calculate a full-rank state-feedback matrix, which usually has smaller gains compared to dyadic state-feedback matrices. The method offers an alternative to the *KNVD* method of CSP, which is based on the Kautsky-Nichols-Van Dooren algorithm [10]. The advantage of the *FullRank* method over *KNVD* method is that, unlike *KNVD*, *FullRank* allows exact and symbolic calculations.

In[36]:= K_{s2} = StateFeedbackGains [Rationalize[aircraft], desiredCLSPoles, Method → FullRank]

Out[36] =

$$\begin{pmatrix} -\frac{1850000}{1277169} & \frac{97}{8838} & \frac{477513423949}{62708997900} & \frac{21391182071767}{9237035390670} & \frac{7540427}{7663014} \\ -\frac{74000}{425723} & \frac{36438463}{7365000} & -\frac{86358369549277}{313544989500000} & \frac{124033390047509}{1847407078134000} & \frac{481697909}{2554338000} \\ -\frac{4910000}{1277169} & 0 & -\frac{1067556541}{212861500} & -\frac{1439371675}{1254179958} & \frac{8224981}{2554338} \end{pmatrix}$$

If, however, calculations are to be carried out numerically, then the *KNVD* algorithm is more robust and therefore preferred. Another potential disadvantage of the *FullRank* method is that the desired complex-conjugate closed-loop poles need to be partitioned compatibly with the controllability indices of the system. Fortunately, this requirement is not a problem for the aircraft system under consideration.

In[37]:= ControllabilityIndices [aircraft]

Out[37]= {2, 1, 2}

Since the controllability indices are given as {2, 1, 2} it is possible to assign up to two pairs of complex-conjugate poles using the FullRank method, which happens to be maximal possible number of complex conjugate poles for the aircraft system. Alternatively, if the controllability indices were, say, {3, 1, 1}, then it is possible to assign only a single pair of complex conjugate poles due to the special structure of the underlying Luenberger controllable canonical form.

Although it allows symbolic calculations, the *FullRank* method does not exploit all of the design freedom available in the pole-assignment problem. For the aircraft example, the state-feedback matrix has $3 \times 5 = 15$ entries. Since only five poles are required to be assigned, there are $15 - 5 = 10$ degrees of freedom in the design. Hence, all constant state-feedback controllers that assign the closed-loop system poles to the required locations can be parameterized as a function of a vector $k_F \in \mathbb{R}^{10}$ [11]. Five components of the vector k_F need to be fixed due to the underlying algorithm used and the special structure of the Luenberger controllable canonical form. PCS fixes the remaining five entries of k_F as well to generate a numeric compensator. This behavior is convenient for novices since it minimizes the complexity of the program interface and avoids confusion. However, we believe it is useful to have options in the *StateFeedbackGains[]* command that yield parameterized forms of pole-assignment compensators. These options can allow additional design criteria such as robustness. A similar comment can be made for *ReducedOrderEstimator[]* and *OutputFeedbackCompensator[]* commands, which are examined in the following sections.

None of the Matlab toolboxes we are aware of use an algorithm such as *FullRank*. The nearest match probably is the *place* command of the Matlab Control Systems Toolbox, which produces a full-rank state-feedback pole assignment matrix using a numerically robust algorithm similar to the KNVD method of CSP.

```
>> Ks2=place(A, B, desiredCLSPoles)
eig(A-B*Ks3)

Ks3 =

   –2.7046   –0.1222   –0.4699    0.4995    3.7182
   –0.6864    6.8830   –0.6432   –0.1819    0.6921
  –10.4168   –0.2709   –6.6626   –2.5597   11.1755

ans =

  –1.0000 + 1.0000i
  –1.0000 – 1.0000i
  –5.0000
  –6.0000
  –7.0000
```

Observer Design

Since all of the states of a system are typically not measured, state feedback is often not feasible. If the system is observable, however, it is possible to reconstruct the states using the knowledge of inputs and outputs of the system (see Figure 1). PCS provides *ReducedOrderEstimator[]* command to find a reduced-order Luenberger observer. The order of such observers depends on the number of states and independent outputs of the system to be observed. The internal dynamics of the observer can be determined arbitrarily as long as the system is observable. An exact reduced-order observer with two real poles at -5 is found in the example below.

In[38]:= observer = ReducedOrderEstimator [Rationalize [aircraft], {–5, –5}]

Out[38]=

$$\left(\begin{array}{cc|cccccc} -5 & 0 & -\frac{63}{40} & 0 & \frac{183}{2500} & -\frac{431820169}{20000000} & \frac{2909}{10000} & \frac{956097}{2500000} \\ 0 & -5 & \frac{4419}{1000} & 0 & -\frac{333}{200} & -\frac{20273677}{4000000} & \frac{97}{2000} & -\frac{5467179}{250000} \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & \frac{1013}{1000} & 0 & \frac{10361}{2500} \\ -1 & 0 & 0 & 0 & 0 & -\frac{43141}{10000} & 0 & \frac{2633}{2500} \end{array}\right)^{\mathcal{S}}.$$

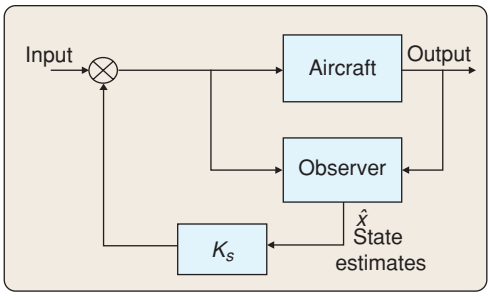


FIGURE 1 Control structure with a state feedback controller and observer. The observer uses the inputs and outputs of the plant to estimate the internal states. A constant state-feedback gain K is then used to place the closed-loop system poles as required.

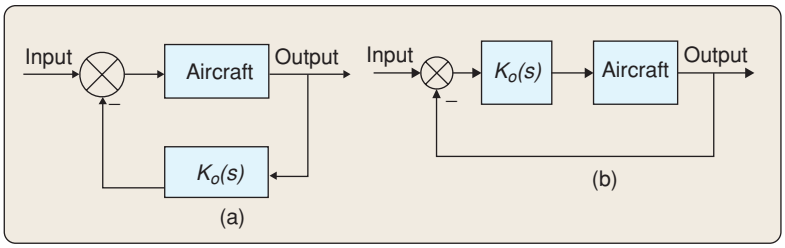


FIGURE 2 Possible output-feedback control structures. (a) Feedback control. (b) Feedforward control. Only measurements are used in the feedback. As far as pole-assignment algorithms are concerned, the structure of the feedback, which affects the zeros of the closed-loop system, is not important. The resulting output-feedback compensator $K_0(s)$ usually has a lower degree compared to a state-feedback controller with a Luenberger observer. The former choice typically entails lower cost and easier implementation.

In the Matlab Control Systems toolbox, it is not possible to obtain a reduced-order state estimator directly. A full-order Luenberger observer, however, is found as follows.

```
>> sys=ss (A, B, C, zeros (3, 3));
L=place(A', C', [–5, –5, –10, –15, –25])';
observer=estim(sys, L);
```

Note that we provide additional observer poles at -10, -15, and -25 since the full-order observer has 3 more poles than the reduced-order observer.

Output-Feedback Pole Assignment

For a controllable and observable least-order system, it is possible to find an output-feedback compensator $K_o(s)$ to place the closed-loop system poles as desired (see Figure 2). The order of the output-feedback compensator needed for arbitrary pole assignment is usually less than the reduced-order observer required for state reconstruction, and is found as follows.

```
In[39]:= OutputFeedbackCompensatorDegree [aircraft]
Out[39]= 1
```

Unless otherwise stated, a dyadic compensator is assumed to be used. That is, the output-feedback compensator is assumed to have the structure $K_0(s) = f_0(s)k_o(s)^T$, where $f_o(s)$ and $k_o(s)^T$ are transfer function vectors of suitable dimensions. It is possible to find the compensator using *OutputFeedbackCompensator[]* command. Note that, since the order of the dyadic output-feedback compensator is one, we must specify an extra desired closed-loop system pole (say -10).

```
In [40]:= K_o1 = OutputFeedbackCompensator [Rationalize [aircraft],
             Join[desiredCLSPoles, {–10}], ControlInput → {0, 1, 0}] // N
```

Out [40]=

−481.764	−2.31542×10^6	218685.	140572.
0.	0.	0.	0.
1.	4829.61	−453.359	−278.615
0.	0.	0.	0.

$\mathcal{S}$

It is possible to check that the closed-loop system poles are located as required.

```
In[41]:= T_o1 = FeedbackConnect [aircraft, K_o1];
         Eigenvalues [T_o1[[1]]]
Out[42]= {–10, –7, –6, –5, –1. + 1.i, –1.–1.i}
```

Using a full-rank (that is, not dyadic) output-feedback compensator, it is usually possible to decrease the order of the compensator even further. For instance, static output feedback is adequate for the aircraft model when a full-rank compensator is to be used.

```
In[43]:= OutputFeedbackCompensatorDegree [aircraft, Method → FullRank]
Out[43]= 0
```

We leave the last desired pole p in symbolic format in the following example to illustrate the symbolic capabilities of PCS (see the bottom of the page).

```
In[44]:= K_o2 = OutputFeedbackCompensator [aircraft], {–1 + I, –1–I, –5, –6, p},
          Method → FullRank];

          T_o2 = Simplify [FeedbackConnect [aircraft, K_o2] ]
```

Out[45]=

0	0	1.132	0	−1	0	0	0
8.24937–2.11609 p	1.p–11.4585	5.54122–1.0905 p	0	0.0705	−0.12	1	0
0	0	0	1	0	0	0	0
77.9252 p–329.915	183.876–36.825 p	40.1576 p–175.763	−0.8556	−1.013	4.419	0	−1.665
27.7737 p–135.271	82.3956–13.125 p	14.3128 p–72.9819	1.0532	−0.6859	1.575	0	−0.0732
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0

$\mathcal{S}$

Note that the closed-loop system description is obtained in terms of the free parameter p. It is possible to pursue further analysis using this symbolic result. The numerator polynomial of the (3, 3) entry transfer function of the system denoted $N_{3,3}(s)$, for example, is calculated as follows:

```
In[46]:= TF = TransferFunction [T_o2] ;
         N_3, 3 = Chop [Numerator [TF [s] [ [3, 3] ] ] ]
Out[47]= 1.665 s^3 – 1.665 ps^2 + 20.1463 s^2 + 40.8219 ps – 197.563 s
         + 245.734 p – 1283.35
```

For $N_{3,3}(s)$ to have all of its roots in the open left-half plane, all the coefficients of this polynomial as well as the first entry r_3 of the third row of the Routh table must be positive. Using this idea, the following commands guarantee that all of the roots of $N_{3,3}(s)$ are in the left shifted half plane $\mathrm{Re}(s) < \sigma$ for stabilizing values of $p < 0$.

```
In[48]:= clst = CoefficientList [N3, 3 / . s → s + σ, s];
         {a0, a1, a2, a3} = clst;
         r3 = – 1/a2 Det [ ( a3  a1 ) ];
                         ( a2  a0 )
         conds = Apply [And, Map [ (# > 0) &, clst ] ] && r3 > 0 && p < 0;
         rcon = Reduce [conds];
         Rationalize [Simplyfy [rcon] ] // N
Out[53]= 9.36354 < σ < 29.5177 ⋀ (σ – 9.36354) (σ + 16.4634) / (σ – 29.5177) < p < 0. ⋁ σ ≥ 29.5177 ∧ p < 0.
```

Note that Mathematica uses $\wedge$ for "logic and," and $\vee$ for "logic or." Hence, the above output shows that it is not possible to have all of the roots of $N_{3,3}(s)$ to the left of the line $\sigma = 9.36354$. This result means that the closed-loop system has at least one right-half-plane zero when the closed-loop system poles are selected as above. Placing the free pole of the closed-loop system at -5 gives us the following compensator:

```
In[54]:= K_o2 / . p → –5
Out[54]=  ( 175.637   –95.6997    92.6986 )
          (  2.24657   4.92073  –0.0410728 )   S
          (  33.993   –32.9997    19.8707 )
```

The roots of $N_{3,3}(s)$ in this case are calculated as follows:

```
In[55]:= s / . Solve [(N_3, 3 / . p → –5) = 0.0]
Out[55]= {–24.4441, –5., 12.3442}
```

Note that the gains of the full-rank compensator are small compared to those of the dyadic compensator, which is usually considered an advantage, since smaller gains usually imply smaller control energy. A possible disadvantage of using the *FullRank* method is that it requires a special partitioning of the desired set of complex conjugate poles similar to the *FullRank* method for the state-feedback case; see the user manual of PCS and [12] for details.

We have discovered that the *OutputFeedbackCompensator[]* command uses exact arithmetic in internal calculations to find required compensators. The result, however, is presented in numeric format when the input arguments are in numeric format. This approach is convenient since the accuracy of the roots can easily be lost with pole-assignment algorithms if numerical calculations are used.

The only command that allows output-feedback pole assignment in the toolboxes of Matlab we examined is the *pplace* command of the Polynomial Toolbox. The command *pplace* requires a left (right) matrix fraction description of the system together with a subset of desired closed-loop system poles and returns a dynamic feedback compensator given as a right (left) matrix fraction description. This command returns a third-order compensator for the aircraft system, appending three poles (-10, -12, and -15) to the set of desired closed-loop poles.

```
>> [N, D] = ss2lmf (A, B, C);
[Nc, Dc] = pplace(N, D, [desiredCLSPoles, –10, –12, –15] )

Nc =

   –1.5e + 002 – 69s        76 + 16s          0.47 + 0.31s
   –18                      12                –1.8
   –4e+002 – 1.9e + 002s    1.3e+003 + 2.9e+002s    1.2 + 0.86s

Dc =

    17 + s        –0.079    –0.14 – 0.07s
  –9.6 – 0.07s     2        –1.9 – s
    1             –36 – s     0
```

Although it produces high-order compensators, the underlying numerical method of PolyX is apparently quite robust as can be observed from the achieved closed-loop system poles:

```
>> pls=roots (N*Nc+D*Dc)
pls =

 –15.0000
 –12.0000
 –10.0000
 –7.0000
 –6.0000
 –5.0000
 –1.0000 + 1.0000i
 –1.0000 – 1.0000i
```

Input-Output Decoupling

In many practical MIMO systems, it is necessary to decouple inputs and outputs so that each output is controlled from a separate input or set of inputs. It is especially convenient to use the ith input to control the ith output for square

systems (number of inputs equal to number of outputs). The first step in this direction is to pair inputs and outputs suitably. We consider the closed-loop system achieved by a full-rank, state-feedback, pole-assignment compensator for the aircraft system as an illustrative example. Note that although we use a closed-loop system as the starting point in the following examples, the decoupling methods discussed here can also be used with the open-loop system.

```
In[56]:= Ts2 = StateFeedbackConnect [aircraft, Ks2] // Chop

Out[56]=
                                                          S
 0      0          1.132      0          -1.  |  0      0      0
 0     -5.         1.018      0.210758    0   | -0.12   1.     0
 0      0          0          1.          0   |  0      0      0
 0      0          0        -42.        -13.  |  0      4.419  0     -1.665
 2.    -0.308186  -12.3604   -2.6782     -2.  |  1.575  0     -0.0732
 ---------------------------------------------+---------------------------
 1.     0          0          0          0    |  0      0      0
 0      1.         0          0          0    |  0      0      0
 0      0          1.         0          0    |  0      0      0
```

To determine the interaction between inputs and outputs of the system at a given frequency such as dc, the relative gain array (RGA) of the system is examined:

```
In[57]:= RelativeGainArray [Ts2] // Chop

Out[57]=
 0.0268729      -0.00535695    0.978484
-0.000575528     1.00536      -0.00478142
 0.973703        0             0.0262974
```

Noting that rows and columns of the RGA sum up to one and the entries near to 1 indicate strong interaction between corresponding inputs and outputs, it is best to control output 1 through input 3, output 2 through input 2, and output 3 through input 1 for the given system. Hence, using a static precompensator, the inputs are rearranged as required.

```
In[58]:= P = | 0   0   1 |
             | 0   1   0 | ;
             |-1   0   0 |
Tp = Chop [ SeriesConnect [TransferFunction [P], Ts2]];
```

Note that the third input to the closed-loop aircraft system is multiplied by -1, since output 1 for a step signal to this input yields a negative steady-state value (that is, entry (1, 3) of $T_{s2}(0) < 0$) as can be observed below.

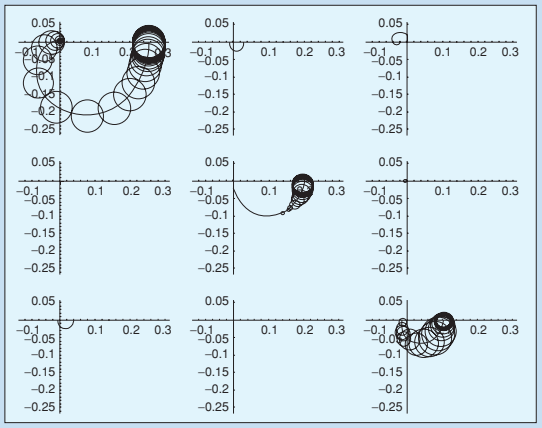
```
In[60]:= TransferFunction [Ts2] [0] // Chop

Out[60]=
-0.018552      0.0308186    -0.254519
-0.00257846    0.2          -0.00807125
 0.105214      0            -0.0396429
```

The *RelativeGainArray[]* command is also used to determine input-output pairings suitable for a given frequency using a second argument. PCS also provides a command called *RelativeGainArrayNumberPlot[]* to determine such pairings in various frequency bands.

The second step in input-output decoupling design is to determine or establish the row (or column) diagonal dominance of the system. A system is *row (column) diagonal dominant* over the bandwidth of interest if, for all rows (columns), the sum of the absolute values of the off-diagonal entries in a row (column) is smaller than the absolute value of the diagonal entry of the same row (column) of the transfer function matrix of the system evaluated at $s = j\omega$ for all frequencies over the same bandwidth. PCS comes with a set of tools for analyzing and designing systems using the concept of diagonal dominance. It is possible, for example, to draw the Nyquist array, inverse Nyquist array (with Gershgorin and Ostrowski circles, if required), column (or row) dominance ratio plots, Perron-Frobenius (PF) eigenvalue/eigenvector plot, singular value plot, and characteristic value plot for system analysis. The Nyquist array, for instance, is an array of Nyquist plots of the entries of the system transfer function matrix. Overlaying the Gershgorin circles, which are centered at the values of the diagonal entries and have a radius given by the sum of the absolute values of all of the off-diagonal entries in the same row (or column), on top of the diagonal Nyquist plots shows the diagonal dominance of the system at first sight:

```
In[61]:= NyquistArrayPlot [Tp, GershgorinCircles → ColumnDominance]
```

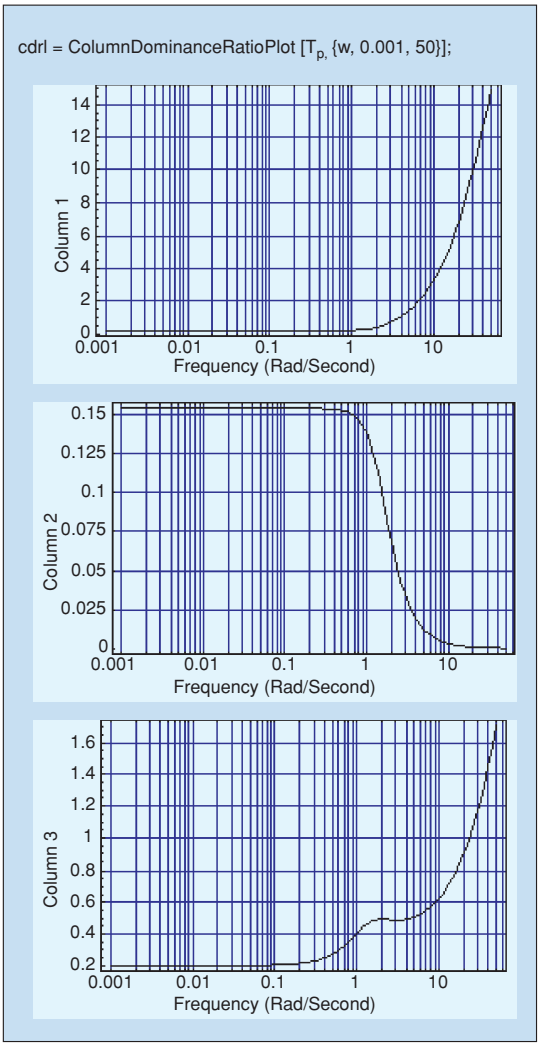


```
Out[61]= -GraphicsArray-
```

The above Nyquist array reveals considerable interaction between the first and third inputs of the system since the Gershgorin circles evaluated through the columns of the transfer function matrix include the origin at high frequencies. The same observation can be made through the column-dominance ratio plot of the system. The column-dominance ratio is defined as the ratio of the sum of the absolute values of all off-diagonal terms in a column to that of the diagonal term in that

column. Obviously, smaller values of the column-dominance ratio indicate less interaction between outputs for each given input, where, for column diagonal dominance, we require the value of column dominance ratio to be less than one.

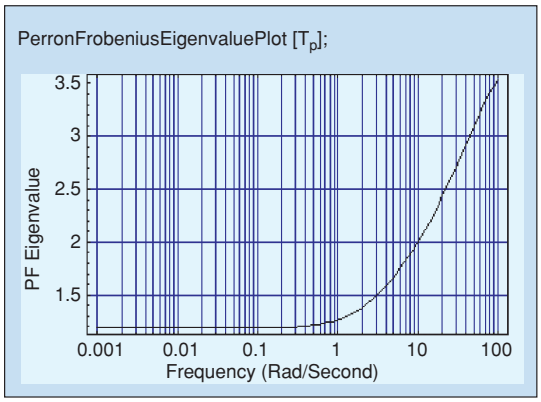
```
In[62]:= cdrl = ColumnDominanceRatioPlot [T_p, {w, 0.001, 50}];
```



Note that column dominance exists for frequencies up to around 4 rad/s for this system.

After determining the diagonal dominance the next design step is to find a precompensator to increase the level of diagonal dominance. Due to its simple structure, using a diagonal input-scaling precompensator is convenient in many cases. It is possible to use the PF eigenvalue plot to determine how much improvement of diagonal dominance can be achieved by employing a constant diagonal precompensator. The best row (column) dominance ratio that can be achieved with such a compensator is given by $\lambda_{pf}(j\omega) - 1$, where $\lambda_{pf}(j\omega)$ is the PF eigenvalue of the transfer function evaluated at a fixed frequency [5].

```
In[63]:= PerronFrobeniusEigenvaluePlot [T_p];
```



The above plot indicates that using a constant diagonal scaling compensator, it is not possible to achieve diagonal dominance for frequencies greater than 10 rad/s, since for higher frequencies the PF eigenvalue is greater than two. If a compensator is to be generated using the Perron-Frobenius theorem, then it is possible to use the *PerronFrobeniusCompensator[]* function of PCS to try to achieve diagonal dominance at a given frequency yielding a static compensator, or a frequency band yielding a dynamic compensator of specified order. For instance, the following produces a Perron-Frobenius compensator with first-order diagonal entries to improve diagonal dominance in the frequency range $\omega \in [0.001, 10]$.

```
In[64]:= PF = PerronFrobeniusCompensator [T_p, {ω, 0.001, 10}, 1]
```

$$\text{Out[64]=}\quad \begin{pmatrix} \dfrac{0.933819\,(s + 1.04323)}{s + 2.19564} & 0. & 0. \\ 0. & \dfrac{0.0546352\,(s + 7.27777)}{s + 2.57933} & 0. \\ 0. & 0. & \dfrac{0.32222\,(s + 6.45839)}{s + 2.36055} \end{pmatrix}^{T}$$

It is possible to check that the compensator given in *Out[68]* does not improve the diagonal dominance of the system much (as predicted), and thus other methods need to be exploited to achieve greater diagonal dominance. Usually a pseudo-diagonalizing compensator, which is a nondiagonal matrix, produces more satisfactory results in terms of achieving diagonal dominance at a given frequency at the cost of a more complex controller structure. Here, the inverse compensator F^{-1} is determined that best diagonalizes the inverse system $Q^{-1} = F^{-1}G^{-1}$ in a least-mean-squares sense. For the system we consider, it is possible to obtain a pseudo-diagonalizing compensator in steady state as follows:

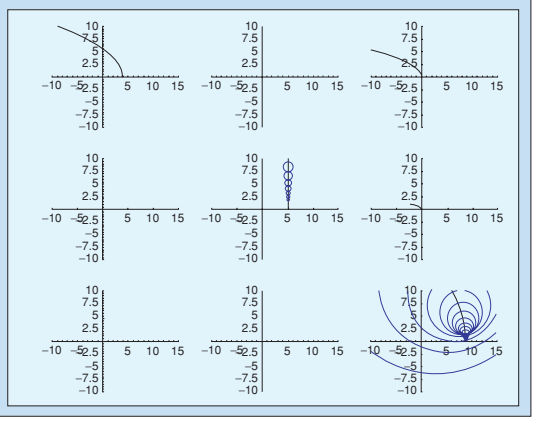
```
In[65]:= InvF = PseudodiagonalizingCompensator [InverseSystem [T_p] ];
         F = Normal [InverSystem [InvF] ]
```

$$\text{Out[66]=}\quad \begin{pmatrix} 0.988208 & -0.118587 & 0.0745845 \\ -0.0446807 & 1.00626 & 0.0104048 \\ -0.372339 & 0.0446813 & 1.04053 \end{pmatrix}^{T}.$$

The *InverseSystem[]* wrapper is used to denote the inverse of a system. Moreover, note that PCS saves

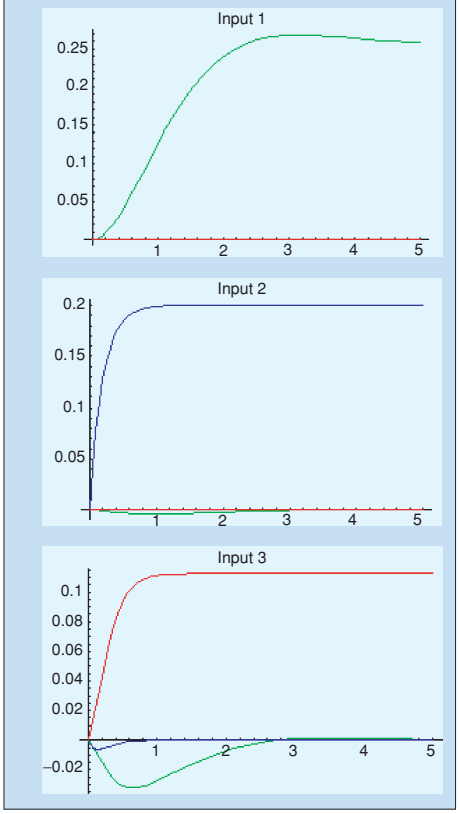
computation time by not actually calculating the inverse of the system until it is necessary, or it is told to do so. The function *InverseSystem[]* can also be used with the *NyquistArrayPlot[]* command to depict an inverse Nyquist array.

```
In[67]:= TF = SeriesConnect [F, TB];
NyquistArrayPlot [InverseSystem [TF], {w, 0.001, 10},
  PlotRange → { {−10, 15}, {−10, 10} }, GershgorinCircles → ColumnDominance,
  CircleStyle → RGBColor [0, 0, 1] ];
```



The above plot reveals that for the system with a pseudodiagonalizing compensator, interactions for first and second inputs are at acceptable levels, whereas a large interaction persists for the third input at high frequencies. This fact can also be observed from the step responses of the system.

```
In[69]:= SimulationPlot [TF, UnitStep [t], {t, 0, 5}, PlotRange → All];
```



In addition to calculating pseudodiagonalizing compensators, PCS also helps in finding high-frequency aligning compensators, which are based on the idea of minimizing the distance between minimum and maximum singular values (depicted by *SingularValuePlot[]*) at a given frequency, as an alternative method for improving diagonal dominance with nondiagonal constant compensators.

After establishing the diagonal dominance of a system, each loop is designed separately using a diagonal output-feedback compensator. Specifically, using the inverse Nyquist array and Gershgorin bands it is usually possible to determine stabilizing gains for such compensators [2]. The characteristic value plot, which is the locus of eigenvalues of the transfer function of a square system evaluated at the borders of the stability region (usually $s = j\omega$), presents an alternative for determining stabilizing diagonal constant output-feedback compensators [13]. PCS provides a command called *CharacteristicValuePlot[]* to draw characteristic values. By using these techniques iteratively it is possible to achieve better decoupling for the given aircraft system. However, our aim here is to demonstrate the use of PCS rather than find the "best" compensator for the given system.

CONCLUSIONS

We have found PCS to be useful and comprehensive. Specially, some of the controller design and synthesis techniques are found to be unique to the toolbox and can be helpful not only for practicing engineers but also for academicians pursuing research. Allowing exact and symbolic calculations for manipulations of polynomial matrices seems to be the greatest advantage of the toolbox. We believe, therefore, that the toolbox can also be used to teach some of the fundamental concepts of multivariable systems to graduate and possibly undergraduate students. We suggest that future improvements to the toolbox include algorithms for analyzing and designing robust control systems such as the techniques found in PolyX.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to Dr. I. Bakshee of Wolfram Research and Prof. M. Sebek and Dr. M. Hromcik of Czech Technical University in Prague for providing complimentary examination copies of PCS and PolyX, respectively, for this review.

REFERENCES

[1] B. Paláncz, Z. Benyó and L. Kovács, "Control system professional suite," *IEEE Control Syst. Mag.*, vol. 25, no. 2, pp. 67–75, Apr. 2005.
[2] N. Munro, *Polynomial Control Systems: User Manual*. Wolfram Research, 2006 [Online]. Available: http://documents.wolfram.com/applications/ pcs/
[3] The Mathworks, Control System Toolbox, [Online]. Available: http://www.mathworks.com/products/control/
[4] PolyX, The Polynomial Toolbox, [Online]. Available: http://www.polyx.com/
[5] H.H. Rosenbrock, *State-Space and Multivariable Theory*. London, U.K.: Nelson, 1970.

[6] J.M. Maciejowski, *Multivariable Feedback Design*. Reading, MA: Addison-Wesley, 1989.
[7] W.M. Wonham, "On pole assignment in multi-input controllable linear systems," *IEEE Trans. Automat. Contr.*, vol. 12, no. 6, pp 660–665, 1967.
[8] M.T. Söylemez and N. Munro, "Pole assignment and symbolic algebra: a new way of thinking," in *Proc. IEE UKACC Control'98*, Swansea, UK, 1998, pp. 1306–1310.
[9] M.T. Söylemez and İ. Üstoğlu, "Designing control systems using exact and symbolic manipulations of formulae," *Int. J. Contr.*, vol. 79, no. 11, pp 1418–1430, 2006.
[10] J. Kautsky, N.K. Nichols, and P. Van Dooren, "Robust pole assignment in linear state feedback," *Int. J. Contr.*, vol. 41, no. 5, pp. 1129–1155, 1985.
[11] M.T. Söylemez, *Pole Assignment for Uncertain Systems*. Baldock, U.K.: Research Studies Press, 1999.
[12] M.T. Söylemez and N. Munro, "A parametric solution to the pole assignment problem using dynamic output-feedback," *IEEE Trans. Automat. Contr.*, vol. 46, no. 5, pp. 711–723, 2001.
[13] I. Postlethwaite and A.G.J. MacFarlane, *A Complex Variable Approach to the Analysis of Linear Multivariable Feedback Systems*. Berlin, Germany: Springer-Verlag, 1979.

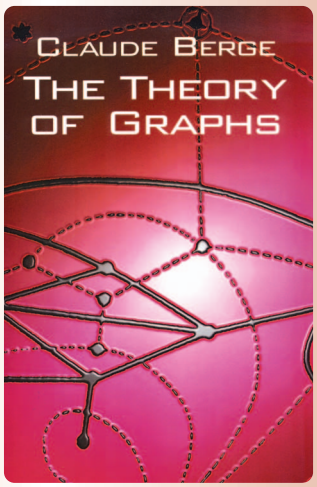
AUTHOR BIOGRAPHIES

Mehmet Turan Söylemez (soylemez@elk.itu.edu.tr) received the B.Sc. degree in control and computer engineering in 1991 from Istanbul Technical University (ITU), Turkey, and the M.Sc. degree in control engineering and information technology from the University of Manchester Institute of Science and Technology (UMIST), U.K., in 1994. He completed his Ph.D. in control engineering in Control Systems Center, UMIST in 1999. He is a professor at the Control and Automation Engineering Division of the School of Electrical and Electronics Engineering of ITU. His research interests include inverse eigenvalue problems, pole assignment, multivariable systems, robust control, computer algebra, numerical analysis, genetic algorithms, modeling and simulation of rail traction systems, PID controllers, and low-order controller design. He is a Member of the IEEE.

İlker Üstoğlu received the B.Sc. degree in electrical engineering from Istanbul Technical University (ITU), Turkey, in 1997 and the M.Sc. degree in control and computer engineering from ITU, in 1999. Since January 1998, he has been working in the Electrical Engineering Department of ITU as a research assistant. His research areas include multivariable systems, robust control, computer algebra, and low-order controller design.

Family Connections

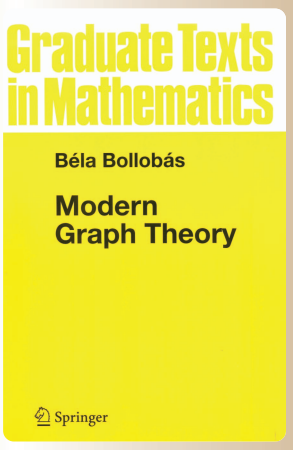
There are many occasions when we use a group of points joined either by lines or by arrows to depict some situation which interests us; the points may stand for people or places or atoms, and the arrows or lines may represent kinship relations or pipelines or chemical bonds. Diagrams like these are met with everywhere, under different names: they are called variously sociograms (psychology), simplexes (topology), circuit diagrams (physics, engineering), organizational structures (economics), communication networks, family trees, etc. D. KONIG was the first person to suggest that the generic name 'graph' be used for all of them, and to undertake a systematic study of their properties.



—From C. Berge, *The Theory of Graphs*, 1966, reprinted by Dover, 2001, p. vii.

Shadows and Light

Graph theory, more than any other branch of mathematics, feeds on problems. There are a great many significant open problems which arise naturally in the subject: many of these are simple to state and look innocent but are proving to be surprisingly hard to resolve. It is no coincidence that Paul Erdos, the greatest problem-poser the world has ever seen, devoted much of his time to graph theory. This amazing wealth of open problems is mostly a blessing, but also, to some extent, a curse. A blessing, because there is a constant flow of exciting problems stimulating the development of the subject: a curse, because people can be misled into working on shallow or dead-end problems which, while bearing a superficial resemblance to important problems, do not really advance the subject.



—From B. Bollobas, *Modern Graph Theory*, Springer, 1998, p. viii.