State-space representation of rational matrices in psi lab

Ramine Nikoukhah

INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1)39 63 55 11

Rapports de Recherche

1 9 9 2

ème anniversaire

N° 1827

Programme 5
traitement du Signal,
Automatique et Productique

STATE-SPACE REPRESENTATION OF RATIONAL MATRICES IN ψLAB

Ramine NIKOUKHAH
François DELEBECQUE

Décembre 1992

RR_1827

State-space representation of rational matrices in Ψlab

Ramine Nikoukhah and François Delebecque

Abstract State-space descriptions have been proven to be effective means for manipulating proper rational matrices. In many systems and control applications, however, rational matrices representing system transfer functions are not necessarily proper. Ψlab uses a state-space description for both proper and improper rational matrices. In this report, we describe this state-space description and the corresponding basic algebraic manipulations.

Représentation d'état des matrices rationelles dans Ψlab

Résumé La manipulation de matrices de fractions rationelles peut s'effectuer de manière efficace en utilisant leur représentation sous forme de variables d'état. Dans les applications pratiques cependant, on rencontre souvent des problèmes conduisant à des matrices rationelles non nécessairement propres. On décrit ici la représentation en variables d'état pour les matrices rationelles non propres qui est implémentée dans Ψlab ainsi que les principaux algorithmes permettant leur manipulation.

1 Introduction

The facilities for manipulating rational matrices (transfer functions) are the most basic elements of every scientific software package designed for systems, control or signal processing applications. It has been known for many years now that basic operations such as concatenation, addition, multiplication, inversion and linear fractional transformations on system transfer functions are much more easier to implement in state-space domain. That is because there exist explicit expressions for the state-space description of the results of these basic operations in terms of the state-space descriptions of their arguments, guarantying "generic" minimality (for example, the order of the state-space description of a product of two state-space descriptions should be equal to or less than the sum of orders of the two state-space descriptions).

State-space descriptions, however, can only realize proper transfer functions (i.e. rational matrices bounded at infinity) and, in many applications, improper transfer functions need to be considered. In this paper, we develop algorithms for performing basic algebraic operations on state-space descriptions of general (polynomial or rational) transfer functions.

There are two ways of representing improper systems in state-space form: one is using descriptor systems [5], i.e., coding the improper transfer function $\Sigma(s)$ into (E, A, B, C) such that

$$\Sigma(s) = C(sE - A)^{-1}B,$$

and second, using state-space descriptions (A, B, C, D) allowing D to be a polynomial matrix [4, 11], i.e.,

$$\Sigma(s) = C(sI - A)^{-1}B + D(s).$$

We will refer to this latter description as the polynomial state-space description (PSSD) to insist on the polynomial nature of the D matrix.

There has been considerable amount of research on descriptor systems in the past (for a survey see [6]). Descriptor systems are very important because they come up naturally in the time-domain formulation of physical systems [10, 2]; the descriptor nature being due to algebraic constraints that often exist in complex models. As a mean of representing improper transfer functions, however, descriptor systems have not been very successful.

In the descriptor formulation of improper systems the polynomial part of the transfer function is coded into a backward nilpotent subsystem which is put in parallel with the standard realization of the proper part (see [10, 7] for realization algorithms). There are a number of difficulties with this formulation. First, the size of the system is larger than the PSSD since all the infinite modes (polynomial part) are coded into the system matrices. Second, this formulation puts together finite and infinite modes which can cause numerical difficulties (in some cases, it is difficult to distinguish a large mode from an infinite mode). To illustrate this point, consider the following (minimal) realization of the polynomial s:

$$E = \begin{pmatrix} 0 & -1 \\ 0 & 0 \end{pmatrix}, A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 \end{pmatrix}.$$

Suppose that due to change of basis and inevitable numerical round-off errors, the matrix

E is perturbed as follows:

$$E = \begin{pmatrix} \epsilon & -1 \\ 0 & \epsilon \end{pmatrix}$$

The perturbed transfer function is $\frac{s}{(1-s\epsilon^2)}$ which for $\epsilon \neq 0$ is a proper system and, in many ways, very different from s. This type of perturbation does not affect so drastically the finite modes of the system. For example $\frac{1}{s+1}$, after perturbation, may become $\frac{1+\epsilon}{s+1+\epsilon'}$, which is not so different from $\frac{1}{s+1}$ provided ϵ and ϵ' are small. In short, even though, theoretically it is useful to treat infinite modes as any other mode, in practice, special care must be taken in handling them. Finally, for converting a descriptor system description into a standard state-space description, to our knowledge, there are no robust algorithms especially if the pencil in question is of high index [2]. And, high index descriptor systems realizing proper systems do come up often as a result of basic operations on minimal descriptor systems. Let us consider multiplying the descriptor realization of s with a minimal descriptor realization of a strictly proper system. The result would be a descriptor system of index 2, realizing a proper system. This means that constructing the correct proper transfer function from its realization is possible only if the non-minimality of this realization can be detected and the realization be reduced to a minimal realization, something which is known to be difficult. This shows again that infinite modes do play a special role and should be treated in a special way.

In the PSSD, the polynomial part of the system is coded into the D matrix and the corresponding algebraic operations are in part strict state-space operations and in part polynomial, the polynomial part corresponding to the infinite modes. We shall see that, as a consequence, infinite modes can be handled much easier since they remain separate from the other modes of the system. Moreover, the realization algorithm for converting an improper transfer function into PSSD is a trivial extension of the standard state-space realization algorithm [4].

We present numerically robust algorithms for performing the basic algebraic manipulations on PSSD's. In particular, in addition to the concatenation and the addition operations which are obtained as trivial extensions of standard state-space operations, we present algorithms for system multiplication, system inversion, and feedback operation.

We start in Section 2 by presenting the concatenation, addition and multiplication operations. In Section 3, we present the inversion and feedback operations and in Section 4, we illustrate the use of these basic operations in Ψlab via a number of examples. We conclude with discussion of the potential applications and future developments in Section 5.

2 Concatenation, addition and multiplication

Let $\Sigma_1 = (A_1, B_1, C_1, D_1(s))$ and $\Sigma_2 = (A_2, B_2, C_2, D_2(s))$ be respectively PSSD's realizing $\Sigma_1(s)$ and $\Sigma_2(s)$. The concatenation operations consist of constructing the PSSD's,

$$\Sigma_r \triangleq \left[\begin{array}{cc} \Sigma_1 & \Sigma_2 \end{array} \right]$$

realizing

$$\Sigma_r(s) = \left[\begin{array}{cc} \Sigma_1(s) & \Sigma_2(s) \end{array} \right]$$

and

$$\Sigma_c \triangleq \begin{bmatrix} \Sigma_1 \\ \Sigma_2 \end{bmatrix}$$

realizing

$$\Sigma_c(s) = \begin{bmatrix} \Sigma_1(s) \\ \Sigma_2(s) \end{bmatrix}.$$

These realizations can easily be constructed as follows:

$$\Sigma_r(s) = \left(\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}, \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix}, \begin{bmatrix} C_1 & C_2 \end{bmatrix}, \begin{bmatrix} D_1(s) \\ D_2(s) \end{bmatrix} \right).$$

and

$$\Sigma_c(s) = \left(\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix}, \begin{bmatrix} D_1(s) & D_2(s) \end{bmatrix} \right).$$

For the addition operation, assuming that Σ_1 and Σ_2 have compatible dimensions, a realization of

$$\Sigma_a(s) = \Sigma_1(s) + \Sigma_2(s)$$

is given by

$$\Sigma_a = \left(\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}, \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \begin{bmatrix} C_1 & C_2 \end{bmatrix}, \begin{bmatrix} D_1(s) + D_2(s) \end{bmatrix} \right).$$

We denote the PSSD addition operation simply with a plus, i.e. $\Sigma_a = \Sigma_1 + \Sigma_2$.

Unlike the two previous cases, the results of standard state-space descriptions cannot trivially be extended to the case of PSSD's as far as system multiplication is concerned, i.e when we like to construct the PSSD Σ realizing

$$\Sigma(s) = \Sigma_1(s)\Sigma_2(s).$$

We denote the PSSD multiplication operation with a star:

$$\Sigma = \Sigma_1 \star \Sigma_2.$$

Theorem 2.1 *Let $\Sigma = \Sigma_1 \star \Sigma_2$. Then a PSSD of Σ, $(A, B, C, D(s))$, can be constructed as follows:*

$$A = \begin{bmatrix} A_1 & B_1C_2 \\ 0 & A_2 \end{bmatrix}, \quad B = Y, \quad C = T,$$

and

$$D(s) = \begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} X(s) + Z(s)Y + D_1(s)D_2(s)$$

where the polynomial matrix $X(s)$ and the constant matrix Y satisfy

$$\begin{bmatrix} B_1D_2(s) \\ B_2 \end{bmatrix} = (sI - A)X(s) + Y$$

and where the polynomial matrix $Z(s)$ and the constant matrix T satisfy

$$\begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} = Z(s)(sI - A) + T.$$

The proof of existence and a method of construction for the matrices Y, T, $X(s)$ and $Z(s)$ are given in Lemma 2.1.

Proof It is straightforward to show that

$$\Sigma(s) = \begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} \left(sI - \begin{bmatrix} A_1 & B_1C_2 \\ 0 & A_2 \end{bmatrix} \right)^{-1} \begin{bmatrix} B_1D_2(s) \\ B_2 \end{bmatrix} + D_1(s)D_2(s).$$

which implies that

$$\Sigma(s) = \begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} \left(sI - \begin{bmatrix} A_1 & B_1C_2 \\ 0 & A_2 \end{bmatrix} \right)^{-1} ((sI - A)X(s) + Y) + D_1(s)D_2(s)$$

which gives

$$\Sigma(s) = \begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} \left(sI - \begin{bmatrix} A_1 & B_1C_2 \\ 0 & A_2 \end{bmatrix} \right)^{-1} Y + \begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} X(s) + D_1(s)D_2(s)$$

which in turn implies

$$\Sigma(s) = T(sI - A)^{-1}Y + Z(s)Y + \begin{bmatrix} C_1 & D_1(s)C_2 \end{bmatrix} X(s) + D_1(s)D_2(s)$$

which proves the theorem. □

Lemma 2.1 *Let $M(s)$ be a polynomial matrix of degree n and A a square constant matrix. Then, if $M(s)$ and A have equal number of rows, there exist a polynomial matrix $N(s)$ of degree $n-1$ and a constant matrix J such that*

$$M(s) = (sI - A)N(s) + J. \tag{2.1}$$

Proof Let

$$M(s) = \sum_{i=0}^{n} M_i s^i$$

and let N_i, $i = 0, ..., n-1$ be obtained from the following recursion

$$\begin{aligned} N_{n-1} &= M_n \\ N_{k-1} &= M_k + AN_k, \quad k = n-1, n-2, ..., 1 \end{aligned}$$

then it is straightforward to see that

$$J = M_0 + AN_0,$$

and

$$N(s) = \sum_{i=0}^{n-1} N_i s^i$$

satisfy (2.1). □

We can of course trivially modify the proof to show that $M(s)$ can also be expressed as $\hat{N}(s)(sI - A) + \hat{J}$ for a polynomial matrix $\hat{N}(s)$ and a constant matrix $\hat{J}$ (in general, $N(s) \neq \hat{N}(s)$ and $J \neq \hat{J}$). Note that this is just a simple Euclidan division. Also note that the constructive proof given for Lemma 2.1 can be used as the basis for an algorithm.

3 System inversion and feedback

There exists an explicit expression for the inverse of state-space-descriptions realizing proper transfer functions with proper inverses, i.e. state-space-description's with invertible D matrix or equivalently having no poles or zeros at infinity. For inverting PSSD's, we do take advantage of this expression, by first regularizing the PSSD to be inverted by multiplying it with a "regularizer" PSSD, "cancelling" its poles and zeros at infinity. We then apply the explicit expression to the regularized PSSD and re-multiply the result with the regularizer PSSD. Even though this procedure may seem simple, there are subtle points that need to be addressed for its successful implementation. One is to prevent the duplication of poles in the process of regularization and de-regularization. And, second, is the a posteriori simplification of the artificial modes introduced by the regularization process. The first problem is solved thanks to the use of the multiplication algorithm presented in the previous section and the second, with an isolation and projection operation presented in Section 3.2.

3.1 Regularization

Definition 3.1 *Let $\Sigma = (A, B, C, D(s))$ have a full (normal) rank transfer function $\Sigma(s)$. Then Σ is called regular if $D(s)$ is constant and has full rank.*

This means that a regular PSSD has no poles or zeros at ∞.

Theorem 3.1 *Let $\Sigma = (A, B, C, D(s))$ have a full column (resp. row) rank transfer function $\Sigma(s)$. Then there exists a square PSSD, Γ, such that*

$$\Sigma_c = \Sigma \star \Gamma \quad (resp.\ \Gamma \star \Sigma)$$

is regular.

In a sense, Γ regularizes Σ by shifting its infinite poles and zeros to other locations in the complex plane, i.e., removing the poles and zeros at infinity (we shall see later that we can choose arbitrarily the location of these shifted poles). We say that Γ is a Σ-regularizer.

Proof Without loss of generality, we shall only consider the case where $\Sigma(s)$ is full column rank. The proof is constructive and can be used as an algorithm for constructing the regularizer Γ.

The regularization process is divided into two stages: first, a PSSD Γ_1 is constructed in such a way that

$$\Sigma_{c,1} = \Sigma \star \Gamma_1$$

is proper (i.e., it removes all the poles at infinity), second, a Γ_2 is constructed removing all the zeros at infinity of $\Sigma_{c,1}$ (i.e., $\Sigma_{c,1} \star \Gamma_2$ having no zeros at infinity). The regularizing matrix is then given by

$$\Gamma = \Gamma_1 \star \Gamma_2.$$

We start by letting

$$\Gamma_1^{(0)} = I$$

and

$$\Sigma^{(0)} = (A^{(0)}, B^{(0)}, C^{(0)}, D^{(0)}(s)) = \Sigma$$

with

$$D^{(0)}(s) = \sum_{k=0}^{d_0} D_k^{(0)} s^k.$$

If $d_0 = 0$ (i.e., $\Sigma^{(0)}$ is proper), we are done and

$$\Gamma_1 = \Gamma_1^{(0)} = I, \quad \Sigma_{c,1} = \Sigma.$$

If $d_0 \neq 0$, find a matrix $W^{(0)}$ which column compresses $D_{d_0}^{(0)}$, i.e.,

$$D_{d_0}^{(0)} W^{(0)} = \begin{pmatrix} X^{(0)} & 0 \end{pmatrix}$$

where $X^{(0)}$ is full column rank and update Γ_1 as follows

$$\Gamma_1^{(1)} = \Gamma_1^{(0)} \star W^{(0)} \star \Lambda^{(0)} \tag{3.2}$$

where

$$\Lambda^{(0)} = \left(0, \begin{pmatrix} I \\ 0 \end{pmatrix}, \begin{pmatrix} I & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}\right)$$

is a PSSD realization of

$$\Lambda^{(0)}(s) = \begin{pmatrix} (1/s)I & 0 \\ 0 & I \end{pmatrix}$$

where the size of the $(1,1)$-block of $\Lambda^{(0)}(s)$ equals the number of columns of $X^{(0)}$. And update $\Sigma_{c,1}$:

$$\Sigma_{c,1}^{(1)} = \Sigma_{c,1}^{(0)} \star W^{(0)} \star \Lambda^{(0)}. \tag{3.3}$$

We can now repeat this procedure until we obtain a proper $\Sigma_{c,1}$ as follows: at stage i,

$$\Sigma_{c,1}^{(i)} = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)}(s))$$

with

$$D^{(i)}(s) = \sum_{k=i}^{d_i} D_k^{(i)} s^k.$$

If $d_i = 0$ (i.e., $\Sigma^{(i)}$ is proper), we are done and

$$\Gamma_1 = \Gamma_1^{(i)}$$

is what removes all the poles at infinity and

$$\Sigma_{c,1} = \Sigma_{c,1}^{(i)}.$$

If $d_i \neq 0$, find a matrix $W^{(i)}$ which column compresses $D^{(i)}_{d_i}$, i.e.,

$$D^{(i)}_{d_i} W^{(i)} = \begin{pmatrix} X^{(i)} & 0 \end{pmatrix}$$

where $X^{(i)}$ is full column rank and update Γ_1 as follows

$$\Gamma_1^{(i+1)} = \Gamma_1^{(i)} \star W^{(i)} \star \Lambda^{(i)}$$

where

$$\Lambda^{(i)} = \left(0, \begin{pmatrix} I \\ 0 \end{pmatrix}, \begin{pmatrix} I & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix}\right)$$

is a PSSD realization of

$$\Lambda^{(i)}(s) = \begin{pmatrix} (1/s)I & 0 \\ 0 & I \end{pmatrix}$$

where the size of the $(1,1)$-block of $\Lambda^{(i)}(s)$ equals the number of columns of $X^{(i)}$. And update $\Sigma_{c,1}$:

$$\Sigma_{c,1}^{(i+1)} = \Sigma_{c,1}^{(i)} \star W^{(i)} \star \Lambda^{(i)}.$$

Clearly, this recursion ends in a finite number of iterations, because, at each iteration the degree d_i of the polynomial part of $\Sigma_{c,1}^{(i)}$ decreases by one.

Now that we have constructed $\Sigma_{c,1}$ and Γ_1, we can start the construction of Σ_c and Γ_2. We proceed as in the previous case by initializing

$$\Gamma_2^{(0)} = I$$

and

$$\Sigma_c^{(0)} = (A^{(0)}, B^{(0)}, C^{(0)}, D^{(0)}) = \Sigma_{c,1}.$$

If $D^{(0)}$ is full rank, we are done and

$$\Gamma_2 = \Gamma_2^{(0)} = I, \quad \Sigma_c = \Sigma_{c,1}.$$

If not, find a matrix $W^{(0)}$ which column compresses $D^{(0)}$, i.e.,

$$D^{(0)} W^{(0)} = \begin{pmatrix} X^{(0)} & 0 \end{pmatrix}$$

where $X^{(0)}$ is full column rank and update Γ_2 as follows

$$\Gamma_2^{(1)} = \Gamma_2^{(0)} \star W^{(0)} \star \Lambda^{(0)}$$

where

$$\Lambda^{(0)}(s) = \begin{pmatrix} I & 0 \\ 0 & sI \end{pmatrix}$$

where the size of the $(1,1)$-block of $\Lambda^{(0)}(s)$ equals the number of columns of $X^{(0)}$. And update Σ_c:

$$\Sigma_c^{(1)} = \Sigma_c^{(0)} \star W^{(0)} \star \Lambda^{(0)}.$$

We can now repeat this procedure until we obtain a Σ_c with no zeros at infinity as follows: at stage i,

$$\Sigma_c^{(i)} = (A^{(i)}, B^{(i)}, C^{(i)}, D^{(i)}).$$

If $D^{(i)}$ is full rank, we are done and

$$\Gamma_2 = \Gamma_2^{(i)}$$

is what removes all zeros at infinity and

$$\Sigma_c = \Sigma_c^{(i)}.$$

If not, find a matrix $W^{(i)}$ which column compresses $D^{(i)}$, i.e.,

$$D^{(i)}W^{(i)} = \begin{pmatrix} X^{(i)} & 0 \end{pmatrix}$$

where $X^{(i)}$ is full column rank and update Γ_2 as follows

$$\Gamma_2^{(i+1)} = \Gamma_2^{(i)} \star W^{(i)} \star \Lambda^{(i)} \tag{3.4}$$

where

$$\Lambda^{(i)}(s) = \begin{pmatrix} I & 0 \\ 0 & sI \end{pmatrix}$$

where the size of the $(1,1)$-block of $\Lambda^{(i)}(s)$ equals the number of columns of $X^{(i)}$. And update Σ_c:

$$\Sigma_c^{(i+1)} = \Sigma_c^{(i)} \star W^{(i)} \star \Lambda^{(i)}. \tag{3.5}$$

This second stage of the algorithm is nothing but Silverman's structure algorithm [9].[1] The algorithm converges in less than n steps where n denotes the order of $\Sigma_{c,1}$. This completes the proof. □

Remarks:

1- All the operations involved in the regularization algorithm are performed in the state-space domain, in particular (3.2), (3.3), (3.4), and (3.5). Note that the result of (3.3) should be a reduction in the degree of the polynomial part of $\Sigma_{c,1}$. Due to numerical round-off errors, however, we may end up with a result having the same polynomial degree as the previous $\Sigma_{c,1}$, the coefficient of the highest degree of its polynomial part being very small. This coefficient should be set to zero otherwise the first stage of the algorithm may never end.

2- For the sake of numerical robustness, in the actual implementation of the algorithm, all the column compression can be performed using orthogonal matrices.

3- We can alter the second stage of the regularization algorithm in such a way as to construct Γ directly (instead of Γ_2) by taking as initial condition $\Gamma_2^{(0)} = \Gamma_1$.

[1] Note that the PSSD multiplication has allowed us to write this algorithm in a simpler fashion.

4- All the finite poles and the zeros of $\Gamma(s)$ as constructed in the above algorithm are at zero. They can however easily be arbitrarily assigned to α. In order to construct a regularizer for Σ having its finite poles and zeros at α, find a regularizer Γ_α for

$$\Sigma_\alpha = ((A + \alpha I), B, C, D(s))$$

having finite poles and zeros at zero. Then

$$\Gamma(s) = \Gamma_\alpha(s - \alpha)$$

is a regularizer for Σ having its finite poles and zeros at α.

The algorithm presented here can be used to study the pole-zero structure at infinity of Σ.

3.2 System inversion

The inversion algorithm for PSSD is somewhat similar to the inversion algorithm presented in [9], at least in the case where the system to be inverted is proper; the main difference is that the result of the inversion operation is put in PSSD form. The two main features of the proposed algorithm are numerical robustness and generic minimality of the result (no duplication or addition of poles).

Definition 3.2 *Let Σ be a square PSSD with invertible transfer function $\Sigma(s)$. Then the PSSD, Σ_i, is called the inverse of Σ if*

$$\Sigma(s)\Sigma_i(s) = I$$

where $\Sigma_i(s)$ is the transfer function of Σ_i.

Lemma 3.1 *Let Σ be a square PSSD with invertible transfer function $\Sigma(s)$ and Γ its regularizer (i.e., $\Sigma \star \Gamma$ regular). Then, Σ_i, the inverse of Σ, can be constructed as follows:*

$$\Sigma_i = \Gamma \star \Delta \tag{3.6}$$

where Δ is the inverse of $\Sigma \star \Gamma$.

Proof Let $\Delta(s)$, $\Sigma_i(s)$ and $\Gamma(s)$ denote respectively the transfer functions associated with Δ, Σ_i and Γ. Then

$$\Sigma_i(s) = (\Gamma(s)\Delta(s))^{-1}$$

which thanks to

$$\Delta(s) = (\Sigma(s)\Gamma(s))^{-1}$$

implies

$$\Sigma_i(s) = \Sigma(s)^{-1}$$

and the lemma is proved. □

Note that all the finite poles and zeros of the regularizer Γ should be cancelled out by the poles and zeros of Δ in (3.6). Pole-zero cancellations are in general very difficult numerical problems. If we multiply two PSSD's having poles at α using the PSSD multiplication algorithm, we end up with a PSSD with poles at α. If α poles should theoretically cancel out by zeros, this realization would be non-minimal (α poles would either be uncontrollable or unobservable). Constructing minimal realizations from non-minimal realizations can be a very difficult numerical problem.

In this case, however, by taking advantage of the fact that we know, a priori, what cancellations should be performed, we can easily overcome this problem. Let us say that all the finite poles and zeros of Γ are at zero, and that $\Gamma \star \Delta$ should not contain any poles at zero. In that case, we can isolate the nilpotent subsystem of the result and eliminate it (since we know a priori that it cannot contribute to the result). Of course, to apply this method to the inversion problem, we should make sure that Δ has no other poles at zero beside those that are destined to be cancelled[2] by Γ.

This result suggests the following inversion algorithm for a PSSD $\Sigma = (A, B, C, D(s))$:

1- Pick an α making sure it is not (and is far from) a zero of Σ. This can be done by examining the full rankedness of the system matrix of Σ evaluated at α:

$$\begin{pmatrix} -\alpha I + A & B \\ C & D(\alpha) \end{pmatrix}$$

which can be done by computing its singular values.

2- Construct the regularizer Γ_α having finite poles and zeros at zero for

$$\Sigma_\alpha = ((A - \alpha I), B, C, D(s + \alpha))$$

and the corresponding regularized system

$$\Sigma_c = \Sigma_\alpha \star \Gamma_\alpha.$$

3- Construct, Δ, the inverse of the regular system $\Sigma_c = (A_c, B_c, C_c, D_c)$ as follows

$$\Delta = ((A_c - B_c D_c^{-1} C_c), B_c D_c^{-1}, -D_c^{-1} C_c, D_c^{-1}),$$

and compute

$$\Sigma_{\alpha,i} = \Gamma_\alpha \star \Delta.$$

4- Eliminate the poles at zero of

$$\Sigma_{\alpha,i} = (A_{\alpha,i}, B_{\alpha,i}, C_{\alpha,i}, D_{\alpha,i}(s))$$

as follows:

[2] More specifically those that come from the inverse of Γ.

i- Put $A_{\alpha,i}$ into ordered Schur form:

$$U^T A_{\alpha,i} U = \begin{pmatrix} A^u_{00} & A^u_{12} \\ 0 & A^u_{22} \end{pmatrix}$$

where A^u_{00} is upper triangular with zeros on the diagonal and A^u_{22} is upper quasi-triangular with non-zero entries on the diagonal. Let Q be the matrix made of the k first columns of U where k is the size of A^u_{00}.

ii- Put $A_{\alpha,i}$ into ordered Schur form:

$$V^T A_{\alpha,i} V = \begin{pmatrix} A^v_{00} & A^v_{12} \\ 0 & A^v_{22} \end{pmatrix}$$

where A^v_{00} is upper quasi-triangular with now non-zeros entries on the diagonal and A^v_{22} is upper triangular with zero entries on the diagonal (clearly the size of $A^v_{22} = k$). Let M be the matrix made of the k last rows of V^T.

iii- Let $E = MQ$ and compute

$$N = ME^{-1}.$$

iv- The reduced PSSD $\Sigma_{\alpha,i,r} = (A_{\alpha,i,r}, B_{\alpha,i,r}, C_{\alpha,i,r}, D_{\alpha,i,r}(s))$ is given by:

$$A_{\alpha,i,r} = NA_{\alpha,i}Q,$$

$$B_{\alpha,i,r} = NB_{\alpha,i},$$

$$C_{\alpha,i,r} = C_{\alpha,i}Q,$$

and

$$D_{\alpha,i,r}(s) = D_{\alpha,i}(s).$$

5- Finally, the inverse of $\Sigma = (A_i, B_i, C_i, D_i(s))$ is obtained as follows:

$$A_i = A_{\alpha,i,r} + \alpha I,$$

$$B_i = B_{\alpha,i,r},$$

$$C_i = C_{\alpha,i,r},$$

and

$$D_i(s) = D_{\alpha,i,r}(s - \alpha).$$

3.3 Feedback

The inversion operation presented in the previous section can be used in the construction of generically minimal realizations for feedback systems. Consider the following feedback system:

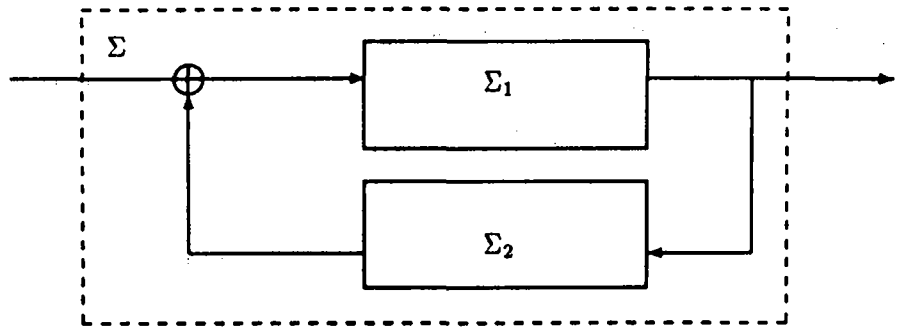

Figure 1: Feedback operation

We like to construct a generically minimal PSSD of the closed-loop transfer function depicted in Figure 1, i.e. a PSSD Σ of the transfer function:

$$\Sigma(s) = \Sigma_1(s)(I - \Sigma_2(s)\Sigma_1(s))^{-1}$$

where Σ_1 and Σ_2 are appropriately dimensioned PSSD's.

Theorem 3.2 *The PSSD of the closed-loop transfer function Σ can be expressed as follows:*

$$\Sigma = \begin{pmatrix} I & 0 \end{pmatrix} \star \Psi \star \begin{pmatrix} 0 \\ I \end{pmatrix}$$

where Ψ is the inverse of

$$\begin{pmatrix} I & -\Sigma_1 \\ -\Sigma_2 & I \end{pmatrix}.$$

The proof is straightforward and is omitted.

This idea can be used to construct fractional compositions for PSSD's as well. In particular, consider the following system:

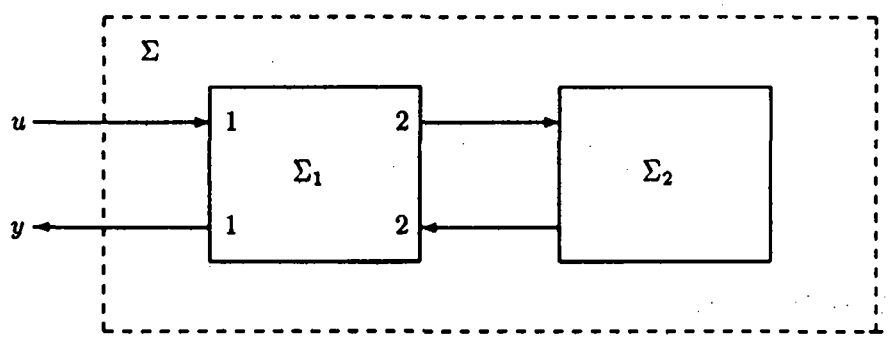

Figure 2: Fractional composition $\Sigma = \Sigma_1 \circ_f \Sigma_2$

where, using the same notations as in [1], we denote the transfer function from u to y as:

$$\Sigma = \Sigma_1 \circ_f \Sigma_2.$$

Theorem 3.3 *The cascade composition Σ can be expressed as follows:*

$$\Sigma = \begin{pmatrix} 0 & 0 & I & 0 \end{pmatrix} \star \Psi \star \begin{pmatrix} 0 \\ 0 \\ 0 \\ I \end{pmatrix}$$

where Ψ is the inverse of

$$\begin{pmatrix} [\Sigma_1] & -I & 0 \\ 0 & -I & 0 & \Sigma_2 \\ I & 0 & 0 & 0 \end{pmatrix}.$$

We can also extend this approach to the case of general cascade compositions:

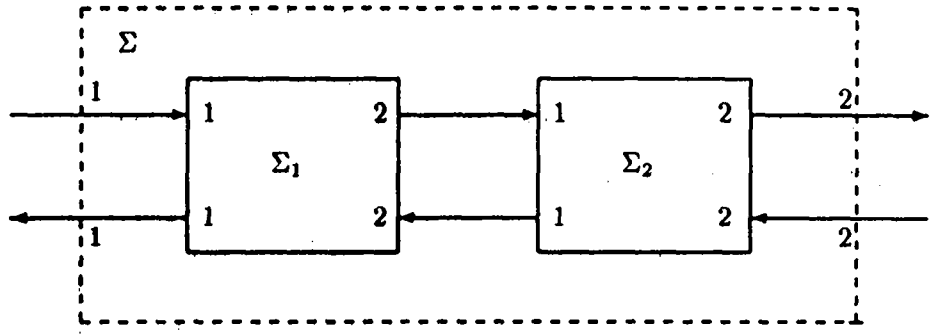

Figure 3: General fractional composition $\Sigma = \Sigma_1 \circ_g \Sigma_2$

Theorem 3.4 *The general cascade composition*

$$\Sigma = \Sigma_1 \circ_g \Sigma_2.$$

can be expressed as follows:

$$\Sigma = \begin{pmatrix} 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \end{pmatrix} \star \Psi \star \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ I & 0 \\ 0 & I \end{pmatrix}$$

where Ψ is the inverse of

$$\begin{pmatrix} [\Sigma_1] & 0 & -I & 0 & 0 \\ 0 & 0 & -I & 0 \\ 0 & -I & 0 & 0 & [\Sigma_2] \\ 0 & 0 & -I & 0 \\ I & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I \end{pmatrix}.$$

4 Ψlab examples

In Ψlab, linear systems can be represented both in state-space domain and in frequency domain. In frequency domain, the representation is a rational matrix representation; it does not care whether the system is proper or not. In state-space domain, linear systems are represented as a list which include matrices A, B, C and D where D is polynomial when the system is not proper. **ss2tf** converts state-space representations into frequency domain representations and **tf2ss** does the opposite. Basic operations on linear systems can be performed in both representations.

4.1 Example 1

In this example, we show how the **inv** operation can be used for inverting an improper system in state-space representation.

```
-->sl=[s^3/(s^2+1),1/s^2;s/(s+5)^3,1/(s+9)]
 sl        =

!      3                              !
!     s                      1        !
!    ---                    ---       !
!        2                   2        !
!   1 + s                   s         !
!                                     !
!            s                  1     !
!        --------            ------   !
!                    2   3            !
!   125 + 75s + 15s + s     9 + s     !

-->sls=tf2ss(sl)
 sls       =

       sls>1

 lss

       sls>2

        columns    1 to   5


! - 0.4873365    0.0413218  - 0.1235074  - 0.8634475    0.          !
! - 0.6793348  - 4.5133291  - 4.3974687    0.          0.          !
! - 21.587479  - 2.713133   - 10.071282  - 14.382389    18.24199    !
!   4.484158     0.1458653    1.3035887    2.5469642  - 2.6110217  !
!   2.5583579  - 0.0917425  - 0.5846735    1.8743175  - 2.4278863  !
```

```
! - 0.9806236    0.1276371  - 0.2210021  - 0.7425117    0.8790051 !
!   1.8361147    0.553016     1.0561388    1.1680004  - 1.4210677 !
! - 141.48577    10.546508  - 35.261487  - 97.133595    121.77543 !

        columns    6 to   8


!   0.           0.           0.        !
!   0.           0.           0.        !
!   0.           0.           0.        !
! - 0.0710749    0.           0.        !
! - 0.0618404    0.0037520  - 0.0615932 !
!   0.1080259    0.9890481    0.        !
! - 0.0203185  - 0.1077599  - 0,0165151 !
!   0.3460646    0.6540799  - 9.0473967 !

       sls>3

!   0.0764506    0.        !
!   0.0504134  - 7.7629441 !
!   1.7017691  - 7.9089505 !
! - 0.2841575    0.8483263 !
! - 0.2007232  - 0.9410664 !
!   0.0779034  - 0.0475738 !
! - 0.1391978  - 0.0864830 !
!   11.088242    1.2530747 !

       sls>4

! - 13.080338    0.           0.    0.    0.    0.    0.    0. !
!   0.0849451  - 0.1288171    0.    0.    0.    0.    0.    0. !

       sls>5

!   s     0 !
!           !
!   0     0 !

       sls>6

!   0. !
!   0. !
!   0. !
!   0. !
!   0. !
```

```
!   0. !
!   0. !
!   0. !

       sls>7

     []

-->sli=inv(sls);

-->sli
 sli       =


       sli>1

 lss

       sli>2


        columns    1 to   5


! - 4.5159212    11.283991  - 79.322447    16.961653  - 5.4619968 !
! - 0.0697923  - 6.079043     39.865564  - 8.3997879    1.7675075 !
!   0.           0.         - 4.3650138    0.8910101  - 0.1462628 !
!   0.           0.           0.         - 0.6066003    3.9063726 !
!   0.           0.           0.           0.         - 0.2972781 !
!   0.           0.           0.           0.           0.2153890 !
!   0.           0.           0.           0.           0.        !
!   0.           0.           0.           0.           0.        !

        columns    6 to   8


! - 108.29098  - 21.442289  - 107.61912 !
!   51.965891    13.249324    52.602475 !
! - 4.9885685  - 0.6739682  - 4.9440812 !
!   15.277895  - 1.5304159    10.262526 !
! - 1.5571005  - 0.4188838    0.0641202 !
!   0.3461348  - 0.7116219  - 0.1781510 !
!   0.           0.5177216    0.3214769 !
!   0.           0.           0.        !
```

```
      sli>3

!   0.0885572    0.3832993 !
!   0.2277712    1.2641279 !
! - 0.1008194    3.726401  !
! - 1.4056255  - 0.1564590 !
!   0.1005046    0.2881162 !
! - 0.0646787    5.8606293 !
!   0.133385     0.6623888 !
!   0.         - 8.6944484 !

      sli>4


       columns    1 to   5


! - 0.0007106  - 0.0046932  - 0.0787692  - 0.6957777    0.0710056 !
! - 0.0811034  - 0.2995485    0.4745526  - 0,0917369    0.0151296 !

       columns    6 to   8


! - 0.0165375    0.0523414  - 0.0267599 !
!   0.5643911    0.2195947    0.5555815 !

      sli>5

!   0    0      !
!                !
!   0    9 + s  !

      sli>6

!   0. !
!   0. !
!   0. !
!   0. !
!   0. !
!   0. !
!   0. !
!   0. !

      sli>7
```

```
c

-->ss2tf(sli)
 ans       =


    column    1
!                2      3     4     5   6   !
!     125s + 75s + 140s + 76s + 15s + s      !
!   ------------------------------------     !
!              2   3      4     5     6   7  !
! - 9 - s - 9s - s + 125s + 75s + 15s + s    !
!                                            !
!                  2   3    4   5            !
!              - 9s - s - 9s - s             !
!   ------------------------------------     !
!              2   3      4     5     6   7  !
! - 9 - s - 9s - s + 125s + 75s + 15s + s    !

    column    2
!                        2      3      4     5   6 !
! - 1125 - 800s - 1335s - 824s - 211s - 24s - s   !
!   ---------------------------------------------!
!            2    3   4      5     6     7   8   !
!   - 9s - s - 9s - s + 125s + 75s + 15s + s     !
!                                                !
!              4      5      6     7   8         !
!       1125s + 800s + 210s + 24s + s            !
!   -------------------------------------        !
!              2   3      4     5     6   7      !
! - 9 - s - 9s - s + 125s + 75s + 15s + s        !

-->pr=sli*sl;

-->ss2tf(pr)
 ans       =

!   1      0   !
!  ---    ---  !
!   1      1   !
!               !
!   0      1   !
!  ---    ---  !
!   1      1   !
```

Next, we consider the fractional composition of improper system **sl** with the improper system $k(s) = 1 + s$ (in state-space), using the Ψlab's **lft** macro.

```
-->k=tf2ss(1+s)
 k          =


       k>1

 lss

       k>2

     []

       k>3

     []

       k>4

     []

       k>5

    1 + s

       k>6

     []

       k>7

 c

-->lft(P,K)
 ans        =


       ans>1

 lss

       ans>2
```

```
        columns    1 to   5

! - 5.0000996    10.292244  - 94.697456    42.930472  - 88.55467   !
!   0.         - 4.9996798    42.155543  - 19.064711    40.703338  !
!   0.         - 1.911D-09  - 5.0002206    2.3858452  - 4.2939111  !
!   0.           0.           0.         - 0.5964716    2.8296072  !
!   0.           0.           0.         - 0.4791401    0.5964716  !
!   0.           0.           0.           0.           0.         !
!   0.           0.           0.           0.           0.         !

        columns    6 to   7

!   5.0992976    2.6536981 !
! - 2.2674607  - 1.3125047 !
!   0.2850521    0.0957825 !
! - 0.0757833    0.1107244 !
! - 0.0731413    0.0139645 !
! - 2.030D-08  - 1.0063274 !
!   0.           2.030D-08 !

       ans>3

! - 10.302351 !
!   4.4909124 !
! - 0.5868213 !
!   0.2207404 !
!   0.0209483 !
!   0.0112575 !
! - 2.271D-10 !

       ans>4

        columns    1 to   5

! - 0.0013913  - 0.0083135  - 0.0215876  - 4.8418823    3.7542574 !

        columns    6 to   7
```

```
!   0.0432993    0.0478583 !

       ans>5

    s

       ans>6

!   0. !
!   0. !
!   0. !
!   0. !
!   0. !
!   0. !
!   0. !

       ans>7

 c

-->ss2tf(ans)
 ans       =

                          2       3          4     5     6   7
    1.125 + 1.25s + 1.25s + 1.25s + 125.125s + 75s + 15s + s
    ---------------------------------------------------------
                          2      3     4     5   6
              125s + 75s + 140s + 76s + 15s + s
```

4.2 Example 2

In this example we use the **inv** operation to compute the inverse of $sE - A$ where E, A is 20×20 pencil of index 5 generated randomly. To do this, we construct a linear system (using **syslin**) in which A, B, and C have dimension 0 and D equals $sE - A$. The **inv** operation in this case, performs as well as an existing specialized routine in Ψlab (generalized Leverrier's algorithm) both in speed and in accuracy.

```
-->E=0*ones(20,20);for i=1:11, E(i,i+1)=1;end

-->E(5,6)=0;E(8,9)=0;E(10,11)=0;

-->E(16:20,16:20)=eye(5,5);

-->A=zeros(20,20);A(1:15,1:15)=eye(15,15);
```

```
-->A22=rand(5,5);

-->A(16:20,16:20)=A22;

-->X=rand(20,20);Y=rand(20,20);E=X*E*Y;A=X*A*Y;

-->sl=syslin('c',[],[],[],s*E-A);

-->sli=inv(sl);

-->spec(sli(2))
 ans       =

!  - 1.9808745              !
!    2.6325485              !
!    1.0533088 + 1.1373242i !
!    1.0533088 - 1.1373242i !
!    0.0551886              !

-->spec(A22)
 ans       =

!    2.6325485              !
!    1.0533088 + 1.1373242i !
!    1.0533088 - 1.1373242i !
!    0.0551886              !
!  - 1.9808745              !

-->w=inv(sli);

-->w-sl
 ans       =


       ans>1

 lss

       ans>2

     []

       ans>3
```

```
[]

  ans>4

[]

  ans>5

      columns    1 to   2

!   2.152D-07 - 9.283D-08s     1.779D-07 - 8.006D-08s  !
!                                                      !
!   6.671D-08 - 4.593D-08s     5.019D-08 - 3.924D-08s  !
!                                                      !
! - 1.538D-07 + 7.682D-08s   - 1.586D-07 + 7.564D-08s  !
!                                                      !
! - 2.082D-07 + 3.657D-08s   - 9.532D-08 + 1.109D-08s  !
!                                                      !
!   0.0000025 - 7.990D-07s     0.0000015 - 5.617D-07s  !
!                                                      !
! - 8.794D-08 + 3.574D-08s   - 1.051D-07 + 4.199D-08s  !
!                                                      !
!   0.0000022 - 7.613D-07s     0.0000014 - 5.706D-07s  !
!                                                      !
!   0.0000011 - 3.330D-07s     6.443D-07 - 2.299D-07s  !
!                                                      !
!   0.0000016 - 4.569D-07s     8.740D-07 - 2.964D-07s  !
!                                                      !
!   0.0000011 - 4.062D-07s     7.207D-07 - 3.112D-07s  !
!                                                      !
! - 1.676D-07 + 6.628D-08s   - 1.051D-07 + 5.010D-08s  !
!                                                      !
!   4.916D-07 - 1.026D-07s     1.578D-07 - 2.905D-08s  !
!                                                      !
! - 0.0000019 + 6.221D-07s   - 0.0000012 + 4.487D-07s  !
!                                                      !
! - 6.507D-07 + 2.383D-07s   - 4.384D-07 + 1.845D-07s  !
!                                                      !
! - 4.024D-07 + 1.243D-07s   - 2.366D-07 + 8.728D-08s  !
!                                                      !
! - 0.0000010 + 3.215D-07s   - 6.189D-07 + 2.313D-07s  !
!                                                      !
! - 0.0000015 + 4.612D-07s   - 8.868D-07 + 3.188D-07s  !
!                                                      !
```

```
!   8.698D-07 - 3.168D-07s      5.768D-07 - 2.464D-07s  !
!                                                       !
!   0.0000011 - 3.565D-07s      6.587D-07 - 2.440D-07s  !
!                                                       !
! - 3.649D-07 + 1.236D-07s    - 2.421D-07 + 9.221D-08s  !

.......
.......
.......

       ans>6

     []

       ans>7

 c
```

5 Conclusion

In this paper, we have presented the alternative state-space description to descriptor systems for representing improper transfer functions used in Ψlab and the algorithms for the corresponding basic algebraic manipulations. The algorithms have been successfully tested on large systems.

This formulation has many applications in control systems engineering and signal processing (construction of standard plant in H_∞ control framework and, singular H_2 and H_∞ problems [3, 8], inner-outer factorization, spectral factorization...).

References

[1] Ball, J. A. and I. Gohberg (1989). Cascade decompositions of linear systems in terms of realizations. *Proc. IEEE Conference on Decision and Control.*

[2] Campbell, S. L. (1980). *Singular Systems of Differential Equations.* Pitman, San Francisco.

[3] Copeland, B. R. and M. G. Safonov (1992). A generalized eigenproblem solution for singular H^2 and H^∞ problems. *Control and Dynamic Systems* 50.

[4] Kailath, T. (1980). *Linear Systems.* Prentice Hall, Englewood Cliffs, NJ.

[5] Luenberger, D. G. (1978). Time-invariant descriptor systems. *Automatica* 14.

[6] Lewis, F. L. (1986). A survey of linear singular systems. *Circuits, Systems and Signal Processing* 5.

[7] Nikoukhah, R. (1988). A deterministic and stochastic theory for boundary value descriptor systems. *Ph. D. Dissertation, M.I.T.*

[8] Nikoukhah, R. and F. Delebecque. On the design of H_∞ controllers when standard assumptions are not satisfied. ***submitted to ACC-93.***

[9] Silverman, L. M. (1969). Inversion of multivariable linear systems. *IEEE Trans. Aut. Control*, AC-14.

[10] Verghese, G. C. (1979). Infinite-frequency behavior in generalized dynamical systems. *Ph. D. Dissertation, Stanford University.*

[11] Wolovich, W. A. (1974). *Linear Multivariable Systems.* Springer-Verlag, New-York.

ISSN 0249-6399