The Use of *Maple* in Computation of Generalized Transfer Functions for Nonlinear Systems

M. Ondera
Institute of Control and Industrial Informatics, Faculty of Electrical Engineering and Information Technology,
Slovak University of Technology in Bratislava, Ilkovičova 3, 812 19 Bratislava, Slovak Republic
Martin.Ondera@stuba.sk

***Abstract* - This paper deals with the recently re-discovered concept of generalized transfer functions of nonlinear control systems. It especially addresses the problems connected with the computation of the transfer functions, which are elements of a fraction field of non-commutative polynomials. An algorithm for calculation of generalized transfer functions for both continuous-time and discrete-time nonlinear systems based on the modified Gauss-Jordan elimination method is presented and its implementation in *Maple* computer algebra system is shown.**

I. Introduction

The concept of generalized transfer functions of nonlinear systems (originally introduced in [12] and later independently re-developed in [6], [7] and [8]) is one of the very recent contributions to modern nonlinear control theory and, as such, is not yet included in traditional textbooks dealing with nonlinear control, e.g. [2], [3], [4]. It is based upon the algebraic approach to nonlinear control summarized in [1] and the theory of skew (i.e. non-commutative) polynomials over the field of meromorphic functions. The generalized transfer functions have many interesting properties and in many ways resemble the traditional linear transfer functions (e.g. the block algebra). However, one of the principal difficulties of the approach rests in a far more complicated computation of the generalized transfer functions, as they are elements of a fraction field of skew polynomials defined over meromorphic functions. Such polynomials are, of course, much more difficult to handle than common ones with real coefficients. Moreover, because of the same reason, specialized software tools, e.g. MATLAB's *Control System Toolbox*, developed for manipulating traditional linear transfer functions cannot be used either. This paper tries to cope with the problem using the *Maple* computer algebra system and its *OreTools* package.

The paper is organized as follows. Section II provides the background necessary for understanding the generalized transfer functions of nonlinear systems and briefly reproduces the main principles of the approach. Section III concentrates on the computational aspects and problems connected with manipulating algebraic objects, such as skew polynomials, fractions of these polynomials and matrices whose elements are fractions of skew polynomials, pointing out some of the differences between the commutative and the non-commutative case. The main attention is dedicated to the algorithm of matrix inversion using modified Gauss-Jordan elimination that plays an important role in the computation of the transfer functions from a state-space description of a nonlinear system. Section IV addresses the implementation problems. It also provides a few illustrative examples. Section V briefly discusses the discrete-time case. Finally, section VI concludes the paper.

II. Generalized Transfer Functions of Nonlinear Systems

Probably all traditional textbooks dealing with nonlinear control (see e.g. [2], [3], [4]) state that there is no such thing as transfer functions of nonlinear systems. The main reason for this conservative opinion is that the Laplace transform, which plays a key role in the theory of linear transfer functions, is not valid for nonlinear systems. However, as was shown in [6], [7], [8] and [12], the Laplace transform is actually not the most crucial for establishing the transfer functions and, in spite of its absence in the nonlinear case, the transfer functions can be defined for a large class of nonlinear systems, too. Moreover, these transfer functions (throughout this paper referred to as the "generalized transfer functions") are in the linear case identical with those derived via Laplace transform. In this section, only the fundamental principles of the generalized transfer functions (and their necessary pre-requisites) will be discussed. For other topics, such as the algebra of the generalized transfer functions or the invariance to regular static state transformations, see [6], [7] or [8]. Similarly, although the generalized transfer functions can be obtained from both state-space and input/output descriptions of nonlinear systems, only the first alternative will be considered in this paper.

A. Algebraic approach to nonlinear control systems

Let us consider a continuous-time nonlinear system described by a system of first-order differential equations of the form

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= h(x,u) \end{aligned} \tag{1}$$

where f and h are meromorphic functions (meromorphic functions are elements of a fraction field of a ring of analytic functions, see [1] for further details), $x \in R^n$, $u \in R^m$ and $y \in R^p$, respectively, denote the state, the input and the output of the system.

Let $\mathcal{K}$ denote the field of meromorphic functions of x, u and a finite number of derivatives of u, i.e. each element of $\mathcal{K}$ is a meromorphic function of the form

$$F\left(x,u,\dot{u},...,u^{(k)}\right);\ k \geq 0 \tag{2}$$

T. Sobh et al. (eds.), Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications, 275–280.

Let us define a derivative operator $\delta : \mathcal{K} \to \mathcal{K}$, such that

$$\begin{aligned} &\delta x_i = \dot{x}_i = f_i(x,u);\ i = 1, ..., n \\ &\delta u_j^{(k)} = u_j^{(k+1)};\ k \geq 0,\ j = 1, ..., m \\ &\delta F\left(x, u^{(k)}\right) = \sum_{i=1}^{n} \frac{\partial F}{\partial x_i} \delta x_i + \sum_{\substack{j=1 \\ k \geq 0}}^{m} \frac{\partial F}{\partial u_j^{(k)}} \delta u_j^{(k)} \end{aligned} \tag{3}$$

and a vector space spanned over $\mathcal{K}$ by differentials of the elements of $\mathcal{K}$, i.e.

$$\varepsilon = \text{span}_{\mathcal{K}} \left\{ d\xi;\ \xi \in \mathcal{K} \right\}. \tag{4}$$

The elements of this vector space (so-called one-forms) are vectors of the form

$$v = \sum_i \alpha_i d\xi_i\ ;\ \alpha_i \in \mathcal{K}. \tag{5}$$

Let us define a differential operator d, acting from $\mathcal{K}$ to ε

$$d : \mathcal{K} \to \varepsilon\ ;\ dF = \sum_{i=1}^{n} \frac{\partial F}{\partial x_i} dx_i + \sum_{\substack{j=1 \\ k \geq 0}}^{m} \frac{\partial F}{\partial u_j^{(k)}} du_j^{(k)} \tag{6}$$

and a derivative operator acting on ε (by abuse of notation also denoted by δ)

$$\delta : \varepsilon \to \varepsilon\ ;\ \delta v = \sum_i \left[\delta(\alpha_i) d\xi_i + \alpha_i d(\delta \xi_i) \right]. \tag{7}$$

Finally, let us also define the recursive use of the derivative operators (3) and (7), i.e.

$$\begin{aligned} &\delta^k F = \delta(\delta^{k-1} F),\ \delta^0 F = F;\ F \in \mathcal{K},\ k \geq 1 \\ &\delta^k v = \delta(\delta^{k-1} v),\ \delta^0 v = v;\ v \in \varepsilon,\ k \geq 1 \end{aligned}. \tag{8}$$

Note: For the sake of further simplification of the notation, the "dot convention" is often used for the derivative operators instead of the δ symbol, that is, e.g. $\delta x_1 = \dot{x}_1$, $\delta^2 u = \ddot{u} = u^{(2)}$, etc.

The equations (1)-(8) form the basics of the algebraic approach to nonlinear systems described in [1] but they are also necessary in order to understand the generalized transfer functions.

B. Some terms from pseudo-linear algebra

Pseudo-linear algebra is the study of common properties of differential and difference operators [5]. Some of its basic terms necessary for understanding the generalized transfer function concept will be explained here.

Let K be a field and $\sigma : K \to K$ an injective endomorphism of K, i.e.

$$\forall\ a, b \in K : \sigma(a+b) = \sigma(a) + \sigma(b) \wedge \sigma(ab) = \sigma(a)\sigma(b) \tag{9}$$

Then a mapping $\delta : K \to K$ satisfying

$$\begin{aligned} &\delta(a+b) = \delta(a) + \delta(b) \\ &\delta(ab) = \sigma(a)\delta(b) + \delta(a)b \end{aligned} \tag{10}$$

is called a *pseudo-derivation*. It is worthy of note that if σ is an identity map, i.e. if $\sigma(a) = a$ for any $a \in K$, then δ is a usual *derivation* acting on K.

The *left skew polynomial ring* given by σ and δ, usually denoted as $K[x;\sigma,\delta]$, is a (non-commutative) ring of polynomials in the indeterminate x over K with the usual addition and the (non-commutative) multiplication given by the commutation rule

$$xa = \sigma(a)x + \delta(a) \tag{11}$$

for arbitrary $a \in K$. Elements of such a ring are called *skew polynomials* or *non-commutative polynomials* [5], [11].

Let V be a vector space over K. A map $\theta : V \to V$ is called *pseudo-linear* if

$$\begin{aligned} &\forall u, v \in V : \theta(u+v) = \theta(u) + \theta(v) \\ &\forall a \in K\ \forall u \in V : \theta(au) = \sigma(a)\theta(u) + \delta(a)u \end{aligned}. \tag{12}$$

Again, let us also define this operation recursively, i.e.

$$\theta^k u = \theta(\theta^{k-1} u),\ \theta^0 u = u;\ u \in V,\ k \geq 1. \tag{13}$$

Skew polynomials can act on the vector space V and thus represent operators. We can define an action

$$\cdot : K[x;\sigma,\delta] \times V \to V;\ \left(\sum_{i=0}^{n} a_i x^i \right) \cdot u = \sum_{i=0}^{n} a_i \theta^i u;\ u \in V. \tag{14}$$

The symbol **.** is usually omitted.

C. Generalized transfer functions

Since the derivative operator δ acting on $\mathcal{K}$ (3) is a pseudo-derivation (with σ being an identity map) and, consequently, the derivative operator δ acting on ε (7) is a pseudo-linear map (σ is again an identity map), we can take advantage of the methods of pseudo-linear algebra (*B.*) and apply them to the one-forms defined in *A.* (see [6] or [7]).

If we consider a left skew polynomial ring over the field of meromorphic functions, with σ being an identity map, i.e. the (non-commutative) ring $\mathcal{K}[s;1,\delta]$, then (14) will turn into

$$\cdot : \mathcal{K}[s;1,\delta] \times \varepsilon \to \varepsilon;\ \left(\sum_{i=0}^{n} a_i s^i \right) \cdot v = \sum_{i=0}^{n} a_i \delta^i v;\ v \in \varepsilon \tag{15}$$

and the commutation rule (11) will be

$$sF = Fs + \dot{F};\ F \in \mathcal{K} \tag{16}$$

As was proven in [6], [7] and [8], the derivative operator δ ((3) and (7), respectively) and the differential operator d (6) are commutable, i.e.

$$\delta^k(dF) = d(\delta^k F) = dF^{(k)} \tag{17}$$

This is a fundamental result, which lets us introduce the generalized transfer functions of nonlinear control systems in the following fashion:

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= h(x,u) \end{aligned} \tag{18}$$

$$\begin{aligned} d\dot{x} &= Adx + Bdu \\ dy &= Cdx + Ddu \end{aligned} \tag{19}$$

where

$$A=\frac{\partial f(x,u)}{\partial x},\ B=\frac{\partial f(x,u)}{\partial u},\ C=\frac{\partial h(x,u)}{\partial x},\ D=\frac{\partial h(x,u)}{\partial u}. \quad (20)$$

Considering the skew polynomial ring $\mathcal{K}[s;1,\delta]$, the mapping (15) and the property (17), we can write (19) as

$$\begin{aligned}(sI-A)dx &= Bdu\\ dy &= Cdx+Ddu\end{aligned} \quad (21)$$

and finally as

$$dy=\left[C(sI-A)^{-1}B+D\right]du=F(s)du \quad (22)$$

where $F(s)$ represents the *generalized transfer function* (or the *generalized transfer matrix* in the MIMO case) of the nonlinear system (18). The expression

$$F(s)=C(sI-A)^{-1}B+D \quad (23)$$

is formally identical with the well-known one that holds for the traditional transfer functions. However, it is important to keep in mind that now the matrices *A*, *B*, *C, D* are not constant but matrices whose elements are meromorphic functions and that the multiplication (15) is non-commutative, i.e. it has to be carried out strictly according to the commutation rule (16) and the order of the terms has to be maintained. As a result, the computation of the inverse matrix to (*sI-A*) is much more complicated (see e.g. [7], [9] or [10]). Besides, further mathematical constructions are necessary to justify (22) and (23) – these will be introduced in the next section.

III. Fractions of Skew Polynomials, Calculating the Generalized Transfer Functions

Since the left skew polynomial ring $\mathcal{K}[s;1,\delta]$ defined in the previous section contains no zero divisors and satisfies the so-called left Ore condition (i.e. each two elements of $\mathcal{K}[s;1,\delta]$ have a common left multiple, see [7], [8], [9], [10] or [11]), it can be embedded to a non-commutative *fraction field* (also known as a *field of fractions* or a *quotient field*) by defining fractions as

$$\frac{a}{b}=b^{-1}.a \quad (24)$$

where $a,b\in\mathcal{K}[s;1,\delta]$ and $b\neq 0$. Addition and multiplication of the fractions of skew polynomials are defined as

$$\frac{a_1}{b_1}+\frac{a_2}{b_2}=\frac{\beta_2 a_1+\beta_1 a_2}{\beta_2 b_1},\ \text{where } \beta_2 b_1=\beta_1 b_2 \quad (25)$$

$$\frac{a_1}{b_1}\cdot\frac{a_2}{b_2}=\frac{\alpha_1 a_2}{\beta_2 b_1},\ \text{where } \beta_2 a_1=\alpha_1 b_2 \quad (26)$$

These are the basic operations that have to be performed with corresponding elements of the matrices involved in (23) in order to calculate the generalized transfer function $F(s)$. Although generally elements of each of the matrices in (23) can be considered fractions of skew polynomials and, therefore, all the individual additions and multiplications can be carried out according to (25) and (26), in fact, there is no need to do everything in the strictly "non-commutative" way – recall that the elements of the matrices *A*, *B*, *C, D* are actually meromorphic functions whose multiplication is commutative (the commutation rule (16) makes a difference only for expressions involving the indeterminate *s*).

Nevertheless, there still remains the problem how to invert the (*sI-A*) matrix, i.e. a matrix whose elements are skew polynomials. Obviously, because of the non-commutative multiplication (15), we cannot use any of the known methods of matrix inversion directly as it was designed for conventional matrices. Besides, not only the multiplication (15) of individual elements is non-commutative; the matrix multiplication itself is non-commutative, too, which brings further difficulties – this "double" non-commutativity e.g. causes that the left inverse matrix is different from the right one (from (21) one can see that we are interested in the left inverse matrix in this case) and the same goes for the determinants. And yet, known methods of matrix inversion can be modified so as to handle the non-commutative multiplication properly.

For example, in [10] linear equations in non-commutative fields are discussed and left- and right-hand determinants are defined. These can be useful in modifying the method of matrix inversion based on a determinant and an adjugate matrix. However, according to our belief, a method which is even more easily adaptable to matrices of skew polynomials is the well-known Gauss-Jordan elimination (see e.g. [13]). The original algorithm requires only slight modifications, the only actual difference being the way how the operations with individual elements are performed in order to get zeros above and below the diagonal. The procedure is illustrated below.

Let us consider a 2nd-order case first. Our task is to calculate the left inverse of the matrix

$$\begin{pmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{pmatrix} \quad (27)$$

where $a_{ij}\in\mathcal{K}[s;1,\delta]$, i.e. a matrix whose elements are skew polynomials. In accordance with the Gauss-Jordan elimination algorithm we will augment the 2nd-order identity matrix to the right of (27), forming the 2×4 block matrix

$$\begin{pmatrix}a_{11} & a_{12} & 1 & 0\\ a_{21} & a_{22} & 0 & 1\end{pmatrix}. \quad (28)$$

From this matrix we need to eliminate the a_{21} and a_{12} elements, i.e. we need to put zeros at the positions of a_{21} and a_{12} by means of elementary row operations. Let us deal with the a_{21} element first – in order to eliminate it we need to find skew polynomials $\beta,\gamma\in\mathcal{K}[s;1,\delta]$ such that

$$\beta a_{11}=\gamma a_{21}. \quad (29)$$

The equation (29) represents the left Ore condition where the value of βa_{11} and γa_{21}, respectively, is the common left multiple of the two elements a_{11} and a_{21} (obviously, in the commutative case, the β and γ would be $\beta=a_{21}$ and $\gamma=a_{11}$). With this done, we can perform the elimination on (28) and get

$$\begin{pmatrix} a_{11} & a_{12} & 1 & 0 \\ 0 & \gamma a_{22} - \beta a_{12} & -\beta & \gamma \end{pmatrix}. \tag{30}$$

In the same way we can eliminate the a_{12} element. The Ore condition corresponding to this case is

$$\varphi(\gamma a_{22} - \beta a_{12}) = \phi a_{12} \tag{31}$$

and the resulting matrix will be as follows:

$$\begin{pmatrix} \phi a_{11} & 0 & \phi + \varphi\beta & -\varphi\gamma \\ 0 & \gamma a_{22} - \beta a_{12} & -\beta & \gamma \end{pmatrix}. \tag{32}$$

Finally, the left inverse of (27) is

$$\begin{pmatrix} \dfrac{\phi + \varphi\beta}{\phi a_{11}} & \dfrac{-\varphi\gamma}{\phi a_{11}} \\ \dfrac{-\beta}{\gamma a_{22} - \beta a_{12}} & \dfrac{\gamma}{\gamma a_{22} - \beta a_{12}} \end{pmatrix}. \tag{33}$$

However, it is often possible to simplify (33) further by cancelling the numerator and the denominator of each of the fractions by their greatest common left divisor.

Let us also sketch a few first steps of the 3rd-order case. The initial augmented matrix is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & 0 & 0 & 1 \end{pmatrix}. \tag{34}$$

In the first step, we will eliminate the a_{21} and a_{31} elements. The corresponding left Ore conditions are

$$\begin{aligned} \beta_2 a_{11} &= \gamma_2 a_{21} \\ \beta_3 a_{11} &= \gamma_3 a_{31} \end{aligned} \tag{35}$$

and the matrix (34) after the elimination will be

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 1 & 0 & 0 \\ 0 & \gamma_2 a_{22} - \beta_2 a_{12} & \gamma_2 a_{23} - \beta_2 a_{13} & -\beta_2 & \gamma_2 & 0 \\ 0 & \gamma_3 a_{32} - \beta_3 a_{12} & \gamma_3 a_{33} - \beta_3 a_{13} & -\beta_3 & 0 & \gamma_3 \end{pmatrix}. \tag{36}$$

Now we shall eliminate the $\gamma_3 a_{32} - \beta_3 a_{12}$ element using the condition

$$\varphi(\gamma_2 a_{22} - \beta_2 a_{12}) = \phi(\gamma_3 a_{32} - \beta_3 a_{12}). \tag{37}$$

The resulting matrix (because of the space limitations the matrix is split and written in two lines) will be

$$\left.\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & \gamma_2 a_{22} - \beta_2 a_{12} & \gamma_2 a_{23} - \beta_2 a_{13} \\ 0 & 0 & \varphi(\gamma_3 a_{33} - \beta_3 a_{13}) - \phi(\gamma_2 a_{23} - \beta_2 a_{13}) \end{pmatrix}\right| \left|\begin{pmatrix} 1 & 0 & 0 \\ -\beta_2 & \gamma_2 & 0 \\ -\varphi\beta_3 + \phi\beta_2 & -\phi\gamma_2 & \varphi\gamma_3 \end{pmatrix}\right. . \tag{38}$$

The $\gamma_2 a_{23} - \beta_2 a_{13}$, a_{13} and a_{12} elements can be eliminated analogically. The final inverse matrix will be a matrix of fractions of skew polynomials whose numerators will be the elements of the right 3×3 block matrix and denominators the corresponding diagonal elements of the left 3×3 block matrix of the 3×6 matrix resulting from the elimination.

IV. Maple Implementation, Illustrative Examples

The *Maple* computer algebra system (version 9.5) provides support for pseudo-linear algebra through its *OreTools* package. The package contains commands for defining and manipulating Ore (i.e. skew, non-commutative) polynomials that simplify the calculation of the generalized transfer functions. Some of the useful commands in the package are:

- *SetOreRing* – define an Ore polynomial ring,
- *LCM, GCD* – compute the least common left or right multiple and the greatest common left or right divisor, respectively, of two or more Ore polynomials,
- *Quotient* – compute the right or left quotient of two Ore polynomials,
- *Add, Minus, Multiply* – add, subtract and multiply, respectively, two Ore polynomials,
- *Convertors[FromPolyToOrePoly]* – convert a polynomial to the corresponding *OrePoly* structure,
- *Convertors[FromOrePolyToPoly]* – convert an *OrePoly* structure to the corresponding polynomial.

On the other hand, the *OreTools* package cannot directly handle fractions of Ore polynomials nor matrices whose elements are fractions of Ore polynomials; therefore, custom procedures have to be programmed for this purpose.

Although, of course, there are no rigid rules as to the implementation of the fractions of Ore polynomials in *Maple*, in our opinion it is advantageous to:

1. represent the fraction as a list with two elements of the *OrePoly* type (the numerator and the denominator),
2. create procedures for the two basic operations with fractions of Ore polynomials, i.e. the addition (25) and the multiplication (26) of the fractions,
3. create a procedure for inverting a matrix of Ore polynomials via the modified Gauss-Jordan elimination method, as described in the previous section,
4. create the main procedure for calculation of a generalized transfer function (23) from a state-space description of a nonlinear system (18) that calls the two above procedures,
5. convert the result (the generalized transfer function) from the *OrePoly* form to the usual transfer-function representation (i.e. fraction of regular polynomials in s).

Because of the space limitations, the *Maple* source code cannot be presented completely. However, some of the procedures are listed in the appendix and the calculation of generalized transfer functions in *Maple* is also illustrated by the examples below.

Example 1: Let us compute the generalized transfer function of the nonlinear system

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix}=\begin{pmatrix}x_2\\ x_1u\end{pmatrix} \qquad (39)$$
$$y=x_1$$

From (20) it follows that

$$A=\begin{pmatrix}0 & 1\\ u & 0\end{pmatrix};\ B=\begin{pmatrix}0\\ x_1\end{pmatrix};\ C=\begin{pmatrix}1 & 0\end{pmatrix};\ D=\begin{pmatrix}0\end{pmatrix} \qquad (40)$$

To be able to calculate the generalized transfer function (23) we need to find the left inverse of the matrix $(sI-A)$. Using the modified Gauss-Jordan elimination algorithm described in the section III we can find out that the left inverse is

$$(sI-A)^{-1}=\begin{pmatrix}\dfrac{s}{s^2-u} & \dfrac{1}{s^2-u}\\ \dfrac{u}{s^2-\dfrac{\dot{u}}{u}s-u} & \dfrac{s-\dfrac{\dot{u}}{u}}{s^2-\dfrac{\dot{u}}{u}s-u}\end{pmatrix} \qquad (41)$$

Finally, the generalized transfer function will be

$$F(s)=\frac{x_1}{s^2-u} \qquad (42)$$

The generalized transfer function can also be computed using a custom procedure (named *TransFunc*) implemented in *Maple*:

```
> f1:=Matrix([[x2],[x1*u]]);
> h1:=Matrix([x1]);
> Fs1:=TransFunc(f1,h1,1);
```

$$f1 := \begin{bmatrix} x2 \\ x1\ u \end{bmatrix}$$

$$h1 := [x1]$$

$$Fs1 := \begin{bmatrix} \dfrac{\mathrm{x1}(t)}{-\mathrm{u}(t)+s^2} \end{bmatrix}$$

Example 2: Let us compute the generalized transfer matrix of the nonlinear MIMO system

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{pmatrix}=\begin{pmatrix}x_3u_1\\ u_1\\ u_2\end{pmatrix} \qquad (43)$$
$$y=\begin{pmatrix}x_1\\ x_2\end{pmatrix}$$

Similarly as in the previous case, we can again take advantage of the *TransFunc* procedure implemented in *Maple*:

```
> f2:=Matrix([[x3*u1],[u1],[u2]]);
> h2:=Matrix([[x1],[x2]]);
> Fs2:=TransFunc(f2,h2,2);
```

$$f2 := \begin{bmatrix} x3\ u1 \\ u1 \\ u2 \end{bmatrix}$$

$$h2 := \begin{bmatrix} x1 \\ x2 \end{bmatrix}$$

$$Fs2 := \begin{bmatrix} \dfrac{\mathrm{x3}(t)}{s} & \dfrac{\mathrm{u1}(t)}{-\dfrac{\left(\dfrac{d}{dt}\mathrm{u1}(t)\right)s}{\mathrm{u1}(t)}+s^2} \\ \dfrac{1}{s} & 0 \end{bmatrix}$$

V. The Discrete-Time Case

Although originally developed for the continuous-time case, the approach can be applied to discrete-time nonlinear systems as well (a comprehensive description is available in [9]). Most of the definitions and properties introduced in the previous sections remain the same, however, certain constructions have to be reformulated in the discrete-time case. These will be mentioned below (the notation used will be slightly different from the one introduced in [9]).

Let us consider a discrete-time nonlinear system described by a system of first-order difference equations of the form

$$\begin{aligned}x(k+1)&=f\big(x(k),u(k)\big)\\ y(k)&=h\big(x(k),u(k)\big)\end{aligned} \qquad (44)$$

where the symbols f, h, x, u, y stand for the same as in the continuous-time case (1). The discrete-time case analogies to (2) and (3) are

$$F\big(x(k),u(k),u(k+1),\ldots,u(k+l)\big);\ l\ge 0 \qquad (45)$$

and $\sigma:\mathcal{K}\to\mathcal{K}$

$$\begin{aligned}&\sigma x_i(k)=x_i(k+1)=f_i\big(x(k),u(k)\big);\ i=1,\ldots,n\\ &\sigma u_j(k)=u_j(k+1);\ j=1,\ldots,m\\ &\sigma F\big(x(k),u(k+l)\big)=F\big(\sigma x(k),\sigma u(k+l)\big);\ l\ge 0\end{aligned} \qquad (46)$$

Note that σ (46) is a shift operator, whereas δ (3) was a derivative operator. The other two constructions that have to be adjusted for use with discrete-time systems are the definitions of the differential operator (6), d: $\mathcal{K}\to\varepsilon$:

$$dF=\sum_{i=1}^{n}\frac{\partial F}{\partial x_i(k)}dx_i(k)+\sum_{\substack{j=1\\ l\ge 0}}^{m}\frac{\partial F}{\partial u_j(k+l)}du_j(k+l) \qquad (47)$$

and the derivative (now shift) operator acting on ε (7):

$$\sigma:\varepsilon\to\varepsilon;\ \ \sigma v=\sum_i\sigma(\alpha_i)d(\sigma\xi_i) \qquad (48)$$

Similarly as in the continuous-time case we will consider a left skew polynomial ring over the field of meromorphic functions $\mathcal{K}$. However, σ will now be the shift operator (46) and $\delta=0$, which can be considered a trivial pseudo-derivation according to (10). Therefore, we can denote the ring as $\mathcal{K}[z;\sigma,0]$, define the $\cdot$ operator (14) as

$$\cdot:\mathcal{K}[z;\sigma,0]\times\varepsilon\to\varepsilon;\ \left(\sum_{i=0}^{n}a_iz^i\right)\cdot v=\sum_{i=0}^{n}a_i\sigma^i v;\ \ v\in\varepsilon \qquad (49)$$

and the commutation rule (11) as

$$zF = \sigma(F)z;\ \ F \in \mathcal{K} \tag{50}$$

These are the most important differences between the continuous- and the discrete-time case, the rest of the constructions either remains unchanged, e.g. (25)–(26), or the necessary modifications are obvious, e.g. (18)–(23). As to the *Maple* implementation, apart from the (optional) change of notations (e.g. z instead of s), the only real difference is the second parameter of the *SetOreRing* command, which has to be *'shift'* in the discrete-time case.

Example 3 (adapted from [9]): Let us compute the generalized transfer function of the discrete-time nonlinear system

$$\begin{pmatrix} x_1(k+1) \\ x_2(k+1) \end{pmatrix} = \begin{pmatrix} x_2(k) + u^2(k) \\ u(k) \end{pmatrix}$$
$$y(k) = x_1(k) \tag{51}$$

We can solve the problem using a custom *Maple* procedure named *TransFuncDisc*:

```
> f1:=Matrix([[x2+u^2],[u]]);
> h1:=Matrix([[x1]]);
> Fz1:=TransFuncDisc(f1,h1,1);
```

$$f1 := \begin{bmatrix} x2 + u^2 \\ u \end{bmatrix}$$

$$h1 := [x1]$$

$$Fz1 := \begin{bmatrix} \dfrac{1 + 2\,\mathrm{u}(k+1)\,z}{z^2} \end{bmatrix}$$

VI. Conclusion

In this paper the problem of computation of generalized transfer functions for nonlinear systems from a state-space description was addressed. The *Maple* computer algebra system and its *OreTools* package were used for the purpose. This is one of the first attempts to implement the generalized transfer functions on a computer in some way but certainly not the last one – in our opinion, computer implementation can stimulate the further development and increase the popularity of the generalized transfer function theory itself. Therefore, our future goals include both the enhancement of the existing *Maple* procedures as well as their migration to *MATLAB*, which is probably a more popular tool within the control engineering community.

Appendix

Two custom *Maple* procedures, *AddOreFractions* and *MulOreFractions*, for addition (25) and multiplication (26), respectively, of two fractions of Ore polynomials are listed below.

```
AddOreFractions:=proc(f1,f2::list)
local beta,beta1,beta2,S;
use OreTools in
   S:=SetOreRing(t,'differential');
   beta:=LCM['left'](f1[2],f2[2],S);
   beta2:=Quotient['right'](beta,f1[2],S);
   beta1:=Quotient['right'](beta,f2[2],S);
   [Add(Multiply(beta2,f1[1],S),
        Multiply(beta1,f2[1],S)),
    Multiply(beta2,f1[2],S)];
end use;
end proc:

MulOreFractions:=proc(f1,f2::list)
local beta,alpha1,beta2,S;
use OreTools in
   S:=SetOreRing(t,'differential');
   if f1[1]<>OrePoly(0) then
    beta:=LCM['left'](f1[1],f2[2],S);
    beta2:=Quotient['right'](beta,f1[1],S);
    alpha1:=Quotient['right'](beta,f2[2],S);
    [Multiply(alpha1,f2[1],S),
     Multiply(beta2,f1[2],S)];
   else
    [OrePoly(0),OrePoly(1)];
   end if;
end use;
end proc:
```

These procedures form an element-oriented basis for the matrix-oriented procedures *InvOreMatrix* (left inverse of a matrix of Ore polynomials) and *TransFunc* (calculation of a generalized transfer function), which are not listed here because of space limitations.

Acknowledgments

This work was supported in part by the Slovak Scientific Grant Agency (VEGA) Project No. 1/3089/06 "Development and integration of methods of the nonlinear systems theory". The author would also like to thank M. Halás for his valuable comments and suggestions.

References

[1] G. Conte, C. H. Moog, and A. M. Perdon, *Nonlinear Control Systems: An Algebraic Setting*. London: Springer, 1999.

[2] A. Isidori, *Nonlinear Control Systems: An Introduction*, 2nd ed. New York: Springer, 1989.

[3] J. J. Slotine, and W. Li, *Applied Nonlinear Control*. New Jersey: Prentice Hall, 1991.

[4] M. Huba, *Nonlinear Systems [in Slovak]*. Bratislava: Vydavateľstvo STU, 2003.

[5] M. Bronstein, and M. Petkovšek, "An introduction to pseudo-linear algebra," *Theoretical Computer Science*, 157, pp. 3–33, 1996.

[6] M. Halás, "Quotients of Noncommutative Polynomials in Nonlinear Control Systems", In: *Proceedings of 18th European Meeting on Cybernetics and Systems Research*, Vienna, Austria, 2006.

[7] M. Halás, and M. Huba, "Symbolic Computation for Nonlinear Systems Using Quotients Over Skew Polynomial Ring", In: *14th Mediterranean Conference on Control and Automation*, Ancona, Italy, 2006.

[8] M. Halás, "An Algebraic Framework Generalizing the Concept of Transfer Functions to Nonlinear Systems," to appear in *Automatica* (provisionally accepted).

[9] M. Halás, and Ü. Kotta, "Extension of the Concept of Transfer Function to Discrete-Time Nonlinear Control Systems, submitted to *European Control Conference 2007*.

[10] O. Ore, "Linear Equations in Non-Commutative Fields," *Annals of Mathematics*, 32, pp. 463-477, 1931.

[11] O. Ore, "Theory of Non-Commutative Polynomials," *Annals of Mathematics,* 34, pp. 480-508, 1933.

[12] Y. Zheng, and L. Cao, "Transfer Function Description for Nonlinear Systems," *Journal of East China Normal University (Natural Science)*, 2, pp. 15–26, 1995.

[13] "Gaussian elimination," "Gauss-Jordan elimination," *Wikipedia, the free encyclopedia*, http://en.wikipedia.org/wiki/Gaussian_elimination.