Abstract: A collection of functions addressing various modelling problems for nonlinear control systems has been developed in Mathematica environment. The mathematical apparatus behind the programs is based on the theory of non-commutative skew polynomial rings. The functions described in paper allow to find the state-space realization of the set of input-output equations, reduce the system equations, compute the transfer matrix of the nonlinear system, check the transfer equivalence of two systems, and finally, the feedforward and feedback compensators can be found such that the compensated systems transfer function coincides with that of the given system.

Keywords: nonlinear control, non-commutative polynomials

1. INTRODUCTION

Over the last decade the theory of non-commutative polynomial rings has been frequently applied in the studies of nonlinear control systems. For that to be possible, the control system, or rather its 'tangent linearized model', has to be described by two polynomial matrices with their elements from the non-commutative ring of Ore polynomials that act on input and output differentials. Polynomial approach has been used to examine numerous modelling, analysis and synthesis problems, for instance, the state-space realization by Belikov et al. [2011b] and by Kotta and Tönso [2012], the system reduction and equivalence by Zheng et al. [2001], Kotta et al. [2006, 2011], extending the concept of transfer function into the nonlinear domain by Halás [2008], Zheng and Cao [1995], computing zero dynamics by Perdon et al. [2007], and solving model matching problem by Halás et al. [2008], Belikov et al. [2011a].

Polynomial method has several advantages, if compared with the earlier method, based on the differential one-forms. The most powerful argument is computation speed - the programs taking advantage of the polynomial methods are able to produce the result remarkably faster than those based only on the approach of the vector spaces of the differential one-forms. Moreover, the program code of the polynomial solution is shorter and more compact, reflecting the fact that polynomial approach allows to express the solutions of the modelling problems via explicit formulas, whereas the approach of one-forms provides only the algorithms for the solution. What is also important, these explicit formulas are remarkably similar to the respective formulas used in the theory of linear systems, except that in the linear case the coefficients of the polynomials are real numbers, the polynomials are applied to the variables $u$ and $y$ rather than their differentials and in the nonlinear case the integration is required, when coming back from the level of polynomials to the level of equations. This brings along the integrability restrictions, because integration is not always possible. This similarity makes the nonlinear system theory easier to understand to the people previously familiar (only) with linear systems.

Several symbolic software packages exist that implement the methods of commutative polynomial theory for control. Two most complete of them are Polynomial Control Systems, written in Mathematica, and Polyx, written for MatLab; both deal with linear systems. Additionally, there exists a small Maple-based package Polycon, see Forsman [1993], which utilizes commutative algebra to handle systems with rational nonlinearities. As for the software related to non-commutative polynomials, the situation is different. In Chyzak et al. [2007] the Maple package OreModules is described, which offers symbolic tools to investigate the structural properties of multidimensional linear systems over Ore algebras. For Maple there is also available a general-purpose Ore algebra package called OreTools, described in Abramov et al. [2005], which does not include any built-in control tools. If comparing the two packages, OreTools seems to us more user friendly and its procedures provide a better basis for applications. Nevertheless, regarding the possible extension of OreModules and OreTools for nonlinear control systems, both have the small but crucial shortage: there is no possibility to define the Ore ring by system equations. In other words, one has to add a procedure for replacing the variables $y_i^{(n_i)}$ in polynomial coefficients by $f_i(\cdot)$, defined by set of equations (2). It is unclear if this replacement can be added upon the package by supplementary procedure, or, what is more...
likely, requires modifying the code of the original package. Finally, there is a small package GTF\_Tools, see Ondera and Haláš [2011], built upon OreTools, implementing the construction of the transfer function of nonlinear system. We are not aware of any other software applying the theory of Ore polynomial rings to nonlinear control problems.

We have developed a collection of Mathematica functions for solving the modelling problems of nonlinear control systems, based on the theory of Ore polynomial rings. The first part of the software includes the functions that implement the basic operations with Ore polynomials, since there is neither built-in functions nor supplement package available for Mathematica, addressing these operations. These basic functions include addition and multiplication, the left(right) quotient and reminder, the greatest common left(right) divisor and the least common left(right) multiple. The second part contains the programs for solving modelling problems by polynomial method.

The above functions are part of our previously developed Mathematica NLControl, devoted to modelling, analysis and synthesis problems of nonlinear control systems Kotta and Tönso [2003], Tönso et al. [2009]. The developed programs are made partly available on NLControl web site and can be found at \url{http://webmathematica.cc.ioc.ee/webmathematica/NLControl/poly}. The main benefit of the web site is that one does not need Mathematica to be installed into local computer, only internet connection and browser are necessary to run the functions.

2. POLYNOMIAL APPROACH IN NONLINEAR CONTROL

Nonlinear control systems can be, in general, described in many ways. In this paper we consider three of them: input-output (i/o) equations, state equations and transfer function. The system may be described, first, by state equations

\[
\dot{x} = f(x, u), \quad y = h(x),
\]

or, second, by i/o equations,

\[
y^{(n)}_{i} = \varphi_{i}(y_{1}, \ldots, y_{i}^{(n_{i}-1)}, u_{i}, \ldots, u_{i}^{(n_{i})}), \quad i = 1, \ldots, p, \quad \nu = 1, \ldots, m, \quad (1)
\]

where \( u = (u_{1}, \ldots, u_{m}) \in U \subset \mathbb{R}^{m} \) input, \( y = (y_{1}, \ldots, y_{p}) \in Y \subset \mathbb{R}^{p} \) output and \( x = (x_{1}, \ldots, x_{n}) \in X \subset \mathbb{R}^{n} \) state variable; \( f, h \) and \( \varphi \) are real analytic functions. The third possibility is to describe the system by its transfer function.

Next, we recall only the basic aspects of the algebraic formalism for nonlinear control systems, described in Conte et al. [2007]. The formalism is adapted for application to i/o equations, since most of the problems treated in this paper assume the system to be given in this form. Let \( K \) denote the field of meromorphic functions in a finite number of the independent system variables \( \{ y_{i}, y_{i}^{(1)}, \ldots, y_{i}^{(n_{i})}, i = 1, \ldots, p, u_{i}, u_{i}^{(1)}, \ldots, u_{i}^{(m_{i})}, \nu = 1, \ldots, m, \nu_{i} \geq 0 \} \). Let \( s : K \to K \) denote the time derivative operator \( \frac{\partial}{\partial t} \). Then the pair \( (K, s) \) is differential field, see Kolchin [1973]. Over the field \( K \) one can define a differential vector space, \( \mathcal{E} := \text{span}_K \{ d\varphi \mid \varphi \in K \} \) spanned by the differentials of the elements of \( K \). Consider a one-form \( \omega \in \mathcal{E} \) such that \( \omega = \sum_{i} \alpha_{i} d\varphi_{i}, \alpha_{i}, \varphi_{i} \in K. \) Its derivative \( \dot{\omega} \) is defined by \( \dot{\omega} = \sum_{i} (\dot{\alpha}_{i} d\varphi_{i} + \alpha_{i} \dot{d}\varphi_{i}) \).

Integrability of the subspace of one-forms can be checked by the Frobenius theorem.

**Theorem 1.** (Choquet-Bruhat et al. [1982].) Let \( \mathcal{V} = \text{span}_K \{ \omega_{1}, \ldots, \omega_{k} \} \) be a subspace of \( \mathcal{E} \). \( \mathcal{V} \) is closed if and only if \( d\omega_{i} \wedge \omega_{1} \wedge \cdots \wedge \omega_{k} = 0 \) for all \( i = 1, \ldots, k \).

Polynomial framework is built upon the linear algebraic framework. The differential field \( (K, s) \) induces a ring of the left differential polynomials \( K[Z; s] \). The elements of \( K[Z; s] \) can be uniquely written in the form

\[
a(Z) = \sum_{i=0}^{n} a_{i} Z^{i}, \quad a_{i} \in K, \quad (3)
\]

where \( Z \) is a formal variable and \( a(Z) \neq 0 \) if at least one of the functions \( a_{i}, i = 0, \ldots, n, \) is nonzero. If \( n_{i} \neq 0 \), then the positive integer \( n \) is called the degree of the left polynomial \( a(Z) \) and may be denoted by \( \text{deg} a(Z) \). The addition of the left polynomials is defined in the standard way. However, for \( a \in K \) the multiplication is defined by the following commutation rule

\[
Z \cdot a := a \cdot Z + s(a). \quad (4)
\]

**Lemma 1.** (Conte et al. [2007]). Let \( F \in K \). Then \( s(dF) = d(Fs) \).

Let us define \( Z^{k} d\nu_{\nu} := (s^{k} y_{\nu}) = d_{\nu}(k) \) and \( Z^{l} du_{\nu} := (s^{l} u_{\nu}) = du_{\nu}(l) \), \( k, l = 1, \ldots, \nu, \nu = 1, \ldots, m \) and \( l \geq 0 \) in the vector space \( \mathcal{E} \). Since every one-form \( \omega \in \mathcal{E} \) has the form

\[
\omega = \sum_{i=1}^{p} \sum_{\nu=0}^{n_{i}} a_{\nu_{i}} d_{\nu_{i}}(i) + \sum_{l=0}^{k} b_{\nu_{i}} Z^{l} du_{\nu},
\]

where \( a_{\nu_{i}}, b_{\nu_{i}} \in K \), so \( \omega \) can be expressed in terms of the left differential polynomials in the following way

\[
\omega = \sum_{\nu=1}^{p} \left( \sum_{i=0}^{n_{i}} a_{\nu_{i}} Z^{i} \right) d\nu_{\nu} + \sum_{l=1}^{k} \left( \sum_{j=0}^{m} b_{\nu_{i}} Z^{l} \right) du_{\nu}.
\]

A left differential polynomial can be considered as an operator acting on vectors \( dy := [dy_{1}, \ldots, dy_{p}]^{T} \) and \( du := [du_{1}, \ldots, du_{m}]^{T} \) from \( \mathcal{E} \) as \( (\sum_{i=0}^{k} a_{i}(Z^{i}) \alpha d\zeta) := \sum_{i=0}^{k} a_{i}(Z^{i}) \alpha d\zeta \) with \( a_{i}, \alpha \in K \) and \( d\zeta \in \{ dy, du \} \). It is easy to note that \( Z(\omega) = s(\omega), \quad \omega \in \mathcal{E}. \)

The nonlinear system (2) can be represented in terms of two polynomial matrices. By differentiating (2) we obtain

\[
dy^{(n_{i})}_{i} - \sum_{\nu=1}^{n_{i}} \sum_{\alpha=0}^{n_{i}} \frac{\partial \varphi_{i}}{\partial y^{(\alpha)}_{\nu}} dy^{(\alpha)}_{\nu} - \sum_{\nu=1}^{n_{i}} \sum_{\beta=0}^{n_{i}} \frac{\partial \varphi_{i}}{\partial u^{(\beta)}_{\nu}} du^{(\beta)}_{\nu} = 0
\]

for \( i = 1, \ldots, p \). Next, using relations \( Z^{\alpha} dy_{\nu} := dy^{(\alpha)}_{\nu}, \quad Z^{\beta} du_{\nu} := du^{(\beta)}_{\nu} \), we can rewrite previous equation as

\[
P(Z) dy + Q(Z) du = 0,
\]

where \( P(Z) \) and \( Q(Z) \) are \( p \times p \) and \( p \times m \)-dimensional matrices respectively, whose elements \( p_{uv}(Z), q_{uv}(Z) \in K[Z; s] \) and
\[ p_{\nu}(Z) = \begin{cases} Z^{n_1} - \sum_{\alpha=0}^{n_\nu} p_{\nu,\alpha} Z^\alpha, & \text{if } \nu = \nu, \\ - \sum_{\alpha=0}^{n_\nu} p_{\nu,\alpha} Z^\alpha, & \text{if } \nu \neq \nu, \end{cases} \]

where \( p_{\nu,\alpha} = \partial \phi_i / \partial y^\alpha \in \mathcal{K} \), \( g_{\nu,\beta} = \partial \phi_i / \partial y^\beta \in \mathcal{K} \).

Equation (5) describes the behavior of system (2) in terms of two polynomial matrices \( P(Z), Q(Z) \) in derivative operator \( K := s \) over the differential field \( \mathcal{K} \).

The ring \( \mathcal{K}[Z; s] \) possesses the left and right division algorithms. If \( p(Z) = p_1(Z)p_2(Z) \) and \( \text{deg} p_1(Z) > 0 \), then \( p_1(Z) \) is called a left divisor of \( p(Z) \). To find the left divisor one may use the left Euclidean division algorithm, see Bronstein and Petkovšek [1996]. The main idea behind this algorithm is that for given two polynomials \( p_1(Z) \) and \( p_2(Z) \) with \( \text{deg} p_1(Z) > \text{deg} p_2(Z) \), there exists a unique left quotient polynomial \( \gamma(Z) \) and a unique left remainder polynomial \( \rho(Z) \) such that \( p_1(Z) = p_2(Z)\gamma(Z) + \rho(Z) \) with \( \text{deg} \rho(Z) < \text{deg} p_2(Z) \). The greatest common left divisor (gcd) of polynomials \( p(Z) \) and \( q(Z) \) can be found by application of left Euclidean algorithm. To find the least common left multiple (lclm), the extended right Euclidean algorithm is used.

The ring \( \mathcal{K}[Z; s] \) can be embedded into a non-commutative quotient field \( \mathcal{K}(Z; s) \) by defining left quotients (or fractions) as

\[ \frac{a(Z)}{b(Z)} := b^{-1}(Z) \cdot a(Z), \quad a(Z), b(Z) \in \mathcal{K}[Z; s]. \]

The addition in \( \mathcal{K}(Z; s) \) is defined by

\[ \frac{a_1(Z)}{b_1(Z)} + \frac{a_2(Z)}{b_2(Z)} = \frac{\beta_2(Z) a_1(Z) + \beta_1(Z) a_2(Z)}{\beta_2(Z) b_1(Z)}, \]

where \( \beta_1(Z), \beta_2(Z) \) are determined by Ore condition \( \beta_2(Z) b_1(Z) = \beta_1(Z) b_2(Z) \). The multiplication of fractions is defined by

\[ \frac{a_1(Z)}{b_1(Z)} \cdot \frac{a_2(Z)}{b_2(Z)} = \frac{a_1(Z) a_2(Z)}{\beta_2(Z) b_1(Z)}, \]

where \( \beta(Z), \beta_2(Z) \) are found from Ore condition \( \beta(Z) a_1(Z) = \alpha_1(Z) b_2(Z) \).

3. FUNCTIONS ON ORE POLYNOMIALS

NLControl uses a special object \texttt{OreRing} to store the information about the Ore ring where polynomials belong to and about the differential field where polynomial coefficients belong. The easiest way to create the object \texttt{OreRing} associated with the control system, is to use the function

\texttt{DefineOreRing[Z, sys]},

where \( Z \) is a polynomial variable and \( \text{sys} \) a control system in the form (1) or (2). It is also possible to work with Ore rings not associated with any control system. In this case, the object \texttt{OreRing} can be created by

\texttt{DefineOreRing[Z, t, TimeDerivative]}.

Additionally, we define the objects representing the Ore polynomial and the fraction of left Ore polynomials. An Ore polynomial in the form (3) is represented as

\[ \text{OreP}[a_n, \ldots, a_1, a_0], \]

where \( a_n, \ldots, a_0 \) are polynomial coefficients. The fraction of Ore polynomials \( p^{-1}(Z)q(Z) \) is entered as

\[ \text{OreR}[p, q], \]

The function \texttt{OreSimplify[p, R]} simplifies the polynomial \( p \), assuming it belongs to the Ore ring \( R \), as described below. The argument \( R \) has to be given as the object \texttt{OreRing}. If there are relations defined between polynomial coefficients, this yields that certain expressions in \( \mathcal{K} \) are equal to zero. The function \texttt{OreSimplify} applies these relations to polynomial coefficients and then simplifies the result. Note that these relations are not applied automatically, since the polynomial object \texttt{OreP} has no information about them.

Addition of polynomials may be performed by \texttt{Mathematica} standard ”+” operator, but addition of polynomial fractions requires a special function

\texttt{OrePlus[OreR[p1, q1], OreR[p2, q2], R]},

where \( p_1, q_1, p_2, q_2 \) are polynomials from the Ore ring \( R \). The function \texttt{OreMultiply[p1, p2, R]} computes a product of polynomials \( p_1, p_2 \) from the Ore ring \( R \) and is based on commutation rule (4). The product of left fractions may be found by the same function. Let \( p \) and \( q \) be polynomials from the Ore ring \( R \). Then the following functions may be applied to them:

- \texttt{LeftQuotientRemainder[p, q, R]} returns a list \( \{\gamma, r\} \), where \( \gamma \) is the left quotient and \( r \) is the left remainder of \( p \) and \( q \).
- \texttt{LeftQuotient[p, q, R]} finds the left quotient of \( p \) and \( q \).
- \texttt{LeftRemainder[p, q, R]} finds the left remainder of \( p \) and \( q \).
- \texttt{LeftGCD[p, q, R]} finds the gcd of \( p \) and \( q \).
- \texttt{LeftLCM[p, q, R]} finds the lclm of \( p \) and \( q \).

Corresponding right-side functions are also available. And finally, there are three functions for matrices with the polynomial entries belonging to the Ore ring \( R \).

- \texttt{OreDot[A1, \ldots, An, R]} finds the product of polynomial matrices \( A_1, \ldots, A_n \) analogously with the standard matrix multiplication function \texttt{Dot}.
- \texttt{LowerLeftTriangularMatrix[A, R]} transforms the rectangular polynomial matrix \( A \) into lower left triangular form.
- \texttt{OreInverse[A, R]} computes the inverse of polynomial matrix \( A \).

4. CONTROL FUNCTIONS

4.1 Control objects

Since version 8.0 \texttt{Mathematica} uses certain data structures or the so-called control objects to represent the control systems. However, these linear control objects cannot be adapted to nonlinear case, because the data used to represent a nonlinear control system have quite a different structure than data describing a linear system. Indeed, nonlinear state equations (1) are usually given by functions \( f \) and \( g \) rather than by matrices over \( \mathbb{R} \), common for linear systems. Besides, linear control objects
do not have possibility to involve the symbols of state (or input) variables. Therefore, we have defined new nonlinear control objects to represent the state equations (1), the i/o equations (2) and the transfer function. The state equations (1) have to be entered in the form

StateSpace[ f, Xt, Ut, t, h, Yt, TimeDerivative ],

where f is a list of the components of the state function; Xt, Ut and Yt define lists of the state, input and output variables, respectively; t is a time argument and h defines the output function. The last argument TimeDerivative indicates continuous-time system. To enter the i/o system (2), one has to use the syntax

IO[eqs, Ut, Yt, t, TimeDerivative ],

where the meanings of the arguments Ut and Yt are the same as in the case of StateSpace and the argument eqs defines the i/o equation (2). The third object is

TransferFunction[Z, F, Ut, Yt, t, TimeDerivative ],

which represents system given by its transfer matrix F with Z being polynomial indeterminate; the rest of the arguments are the same as for i/o equations.

The keywords StateSpace, IO and TransferFunction acts also as functions transforming system between different representation.\(^1\)

4.2 Reduction

A nonlinear control system described by equations (2) is reducible iff the gcld \( G_L(Z) \) of polynomial matrices \( P(Z) \) and \( Q(Z) \) in (5) is non-unimodular matrix (i.e. its inverse is not a polynomial matrix), see Kotta et al. [2006]. In case of the non-unimodular \( G_L(Z) \), equation (5) may be rewritten as

\[
G_L(Z)[\dot{P}(Z)dy - \dot{Q}(Z)du] = 0
\]

and the reduced system equations can be found by integrating the one-forms in the brackets. Linear transformations with the one-forms may be necessary to obtain integrable one-forms. To find the gcld of matrices \( P(Z) \) and \( Q(Z) \) one has to transform the composite matrix \([P(Z)\mid Q(Z)]\) into the lower left triangular form \([G_L(Z)]_0\), then the matrix \( G_L(Z) \) is the gcld of \( P(Z) \) and \( Q(Z) \).

The function Reduction finds, if possible, for the system described by the set of i/o equations (2), a new, lower-order representation, being transfer equivalent to the original set of equations.

Example 1. Consider the system

\[
\dot{y}_1 = u_1 - \dot{y}_1 + \dot{u}_1, \quad \dot{y}_2 = -u_1 - y_2 \dot{u}_1 + \dot{u}_2 + \dot{y}_1 - u_1 \dot{y}_2. \quad (6)
\]

Let us create the respective i/o system in Mathemtica:

\[
\text{In}[1]=\text{eqs} = \{ y''\left[t\right] == u_1\left[t\right]- y'\left[t\right]+u'_1\left[t\right], \\\n& y''_2\left[t\right] == -u_1\left[t\right]-y_2\left[t\right]u'_1\left[t\right]+ u'_2\left[t\right]+y'_1\left[t\right]- u_1\left[t\right]y'_2\left[t\right], \\\n& Ut = \{ u_1\left[t\right], u_2\left[t\right] \}; \quad Yt = \{ y_1\left[t\right], y_2\left[t\right] \}; \\\n& \text{ioeq1} = \text{IO[eqs, Ut, Yt, t, TimeDerivative]};
\]

The following row finds the reduced form of the system:

\[
\text{In}[5]=\text{ioeq2} = \text{Reduction}[\text{ioeq1}]
\]

\[
\text{Out}[5]= -u_1 + y'_1 = 0 \quad \text{and} \quad -u_2 + u_1 y'_2 + y'_2 = 0
\]

Note that the returned i/o equations are internally represented as a control object beginning with IO; however, they are printed to the screen in a more user-friendly form with argument t omitted. Thus, returned systems may be used as an input for the other NLControl functions.

4.3 Realization

The realization problem is to construct the state equations (1) of order \( n = n_1 + \ldots + n_p \) from the set of i/o equations (2), if possible. Note that unlike the linear case, the state-space realization does not exist for every nonlinear i/o model (2). The function Realization is based on Belikov et al. [2011b]. Denote the row vectors of \( P(Z) \) and \( Q(Z) \) by \( p_i(Z) := [p_{i1}(Z), \ldots, p_{ip}(Z)] \) and \( q_i(Z) := [q_{i1}(Z), \ldots, q_{im}(Z)] \), respectively. Define the set of one-forms

\[
\omega_{i,l}(Z) := \begin{bmatrix} y_{i,l}(Z) \\ p_{i,l}(Z) \end{bmatrix},
\]

(7)

for \( i = 1, \ldots, p, \ l = 1, \ldots, n_i \), where \( p_{i,l}(Z) \) and \( q_{i,l}(Z) \) are Ore polynomials, which can be recursively calculated from equalities

\[
p_{i,l-1}(Z) = Z \cdot p_{i,l}(Z) + r_{i,l}, \ \text{deg} r_{i,l} = 0, \quad r_{i,l-1}(Z) = Z \cdot q_{i,l}(Z) + p_{i,l}, \ \text{deg} p_{i,l} = 0,
\]

with initial polynomials \( p_{i,0}(Z) := p_i(Z) \) and \( q_{i,0}(Z) := q_i(Z) \). Then differentials of the state coordinates can be calculated as integrable linear combinations of the one-forms \( \omega_{i,l} \), where \( i = 1, \ldots, p, \ l = 1, \ldots, n_i \). The function Realization checks whether the system given by the set of input-output equations can be transformed into the classical state-space form and in case of the confirmative answer finds the state equations.

Example 2. Let us find the realization of the reduced system obtained in Example 1.

\[
\text{In}[6]=\text{sseq2} = \text{Realization}[\text{ioeq2}, \{ x1[t], x2[t] \}]
\]

\[
\text{Out}[6]= x'_1 = u_1 \quad \text{and} \quad x'_2 = u_2 - u_1 x_2
\]

\[
\text{y}_1 = x_1 \quad \text{and} \quad \text{y}_2 = x_2
\]

Example 3. Consider a single-input single-output (SISO) system \( \dot{y} = \ddot{u} + y \).

\[
\text{In}[7]=\text{eq3} = y''\left[t\right] == u''\left[t\right] y'\left[t\right] + y\left[t\right]; \quad \text{ioeq3} = \text{IO}[^{\text{eq3}, u[t], y[t], t, \text{TimeDerivative}}]; \quad \text{ioeq3} = \text{IO}[^{\text{eq3}, u[t], y[t], t, \text{TimeDerivative}}];
\]

\[
\text{In}[9]=\text{Realization}[\text{ioeq3}, \{ x1[t], x2[t], x3[t] \}]
\]

IntegrateOneForms::nonint: The set of differential oneforms is not completely integrable.

\[
\text{Out}[9]= \{ \}
\]

NLControl has print a message informing about non-integrable subspace; thus the state space form does not exist for this system.

\(^1\) Except that transformations from the transfer matrix representation into the other forms is at moment implemented only for single-output systems.
4.4 Transfer function/matrix

According to Halás [2008], Zheng and Cao [1995], transfer function of the system (2) can be computed as \( F(Z) = P^{-1}(Z)Q(Z) \), where \( F(Z) \) is a matrix with the elements \( f(Z) = b^{-1}(Z)a(Z) \in K(Z,s) \). The crucial step of finding the transfer matrix is computation of the inverse matrix \( P^{-1}(Z) \), which can be performed by modified Gauss-Jordan elimination, see Ondera [2007]. To compute the transfer matrix from the state equations (1) one first has to differentiate the state equations to obtain the polynomial representation of the system:

\[
\frac{dx}{dt} = A dx + B du \\
\frac{dy}{dt} = C dx,
\]

where \( A = \frac{\partial f}{\partial x}, B = \frac{\partial f}{\partial u} \) and \( C = \frac{\partial h}{\partial x} \). The transfer matrix of (1) is \( F(Z) = C(sI - A)^{-1}B \), where \( I \) is identity matrix.

The transfer function/matrix of the nonlinear system may be found by using function \texttt{TransferFunction[Z, sys]}. 

**Example 4.** The transfer matrix of the reduced system in Example 1 is as follows:

\[
\text{In}[10]:= \text{MatrixForm[ tf2=TransferFunction[Z,ioeq2] ]} \\
\text{Out}[10]= \begin{pmatrix} 1 & 0 \\ -Z & 1 \end{pmatrix}
\]

4.5 Transfer equivalence

Two systems are considered to be transfer equivalent, if their transfer functions/matrices are the same. Thereby both matrices are assumed to have in the irreducible form and the polynomial with the highest degree is assumed to be monic. The function \texttt{TransferEquivalent} allows to check the transfer equivalence of control systems described either by i/o or state equations.

**Example 5.** Consider the systems \texttt{ioeq1} and \texttt{ioeq2} in Example 1.

\[
\text{In}[11]:= \text{TransferEquivalent[ioeq1, ioeq2]} \\
\text{Out}[11]= \text{True}
\]

As expected, the system is equivalent with its reduced system.

4.6 Model matching problems

The functions described in this subsection rely on Halás et al. [2008]. At present our package allows to solve model matching problems only for SISO systems, which may be obtained from (2) by taking \( p = m = 1 \) and defining \( \varphi(\cdot) = \varphi_1(\cdot), y := y_1, u := u_1, n := n_1 \) and \( r := r_1 \):

\[
y^{(n)} = \varphi(y, \ldots, y^{(n-1)}, u, \ldots, u^{(r)}).
\]

**Feedforward compensator**

Given the nonlinear systems \( F \) and \( G \), described by their transfer functions

\[
F(Z) = \frac{q_F(Z)}{p_F(Z)} \quad \text{and} \quad G(Z) = \frac{q_G(Z)}{p_G(Z)},
\]

respectively, the task is to find (proper) feedforward compensator \( R \), described by its transfer function \( R(Z) = \frac{p_R^{-1}(Z)q_R(Z)}{p_R(Z)} \) such that the transfer function of compensated system coincides with that of \( G \), as depicted in Fig. 1. The transfer function of the system \( R \) may be computed

\[
\begin{array}{c}
G(Z) \\
\text{feedforward} \\
\text{compensator}
\end{array}
\]

\[
\begin{array}{c}
dv(t) \\
R(Z) \\
du(t) \\
F(Z) \\
dy(t)
\end{array}
\]

Fig. 1. Compensated system

\[ R(Z) = \frac{G(Z)}{F(Z)} = \frac{p_F(Z)}{q_F(Z)} \cdot \frac{q_G(Z)}{p_G(Z)}. \]

The feedforward compensator exists for \( F(Z) \neq 0 \) if the one-form

\[
p_R(Z) du - q_R(Z) dv
\]

is integrable. If the one-form (11) is integrable, its i/o representation may be obtained by integrating the one-form (11).

**Example 6.** Consider the systems \( F : \dot{y} = y + uu \) and \( G : \dot{y} = y - v \). After assigning to the systems the symbols \( Fio \) and \( Gio \), respectively,

\[
\text{In}[12]= \text{Fio} = y''[t] == y[t] + u'[t]*u[t]; \\
\text{Gio} = y''[t] == y'[t] - v[t];
\]

\( Fio = 10[\text{Fio}]; y[t], t, \text{TimeDerivative}; \\
\text{Gio} = 10[\text{Gio}]; y[t], t, \text{TimeDerivative}]; \)

the feedforward compensator may be found as

\[
\text{In}[16]= \text{FeedforwardCompensator[Fio, Gio]}
\]

\( \text{Out}[16]= -v - \dot{u}^2 - \dot{v} + uu = 0 \)

**Feedback compensator**

Given the nonlinear systems \( F \) and \( G \) by their transfer functions (10), the goal is to find the feedback compensator \( R \) to the system \( F \) such that the transfer function of compensated system coincides with that of \( G \), as depicted in Fig 2.

\[
\begin{array}{c}
G(Z) \\
\text{feedback} \\
\text{compensator}
\end{array}
\]

\[
\begin{array}{c}
dv(t) \\
R(Z) \\
du(t) \\
F(Z) \\
dy(t)
\end{array}
\]

Fig. 2. Compensated system

It is required that \( F(Z) \neq 0 \) and without loss of generality we can assume that \( \deg p_G(Z) \geq \deg p_F(Z) \). Then transfer matrix of the compensator \( R \) may be expressed as

\[
R(Z) = \begin{pmatrix} q_{R_L}(Z) & q_{R_R}(Z) \\ p_{R_L}(Z) & p_{R_R}(Z) \end{pmatrix},
\]

where \( q_{R_L}(Z) = q_G(Z) \) and polynomials \( q_{R_R}(Z) \) and \( p_{R_R}(Z) \) can be found from the equality \( p_G(Z) = \gamma(Z)p_F(Z) - q_{R_L}(Z) \) as follows: \( q_{R_L}(Z) \) is the right remainder of \( p_G(Z) \) and \( p_{R}F(Z) \) and \( p_{R}R(Z) = \gamma(Z) : q_{F}(Z) \) with \( \gamma(Z) \) being right quotient of \( p_G(Z) \) and \( p_{R}(Z) \). Note that system \( R \) has two inputs \( v \) and \( y \), and a single output \( u \), thus \( R(Z) \).
is a $2 \times 1$ matrix. The i/o equation of the compensator is obtained by integrating the one-form

$$p_R(Z) \, du = q_{R_y}(Z) \, dv + q_v(Z) \, dy$$  \hspace{1cm} (12)$$

The one-form (12) is always integrable, thus the feedback compensator always exists.

**Example 7.** Computing the feedback compensator for the systems $F$ and $G$ defined in Example 6 yields the following result:

\begin{verbatim}
In[17]:= FeedbackCompensator[Fio, Gio]
Out[17]= -\nu + \nu \ddot{u} - \dot{y} = 0
\end{verbatim}

5. CONCLUSIONS

The paper describes the sub-package of *Mathematica*-based software NLControl, which allows to solve several modelling problems for nonlinear control systems and mathematically relies on the formalism of non-commutative polynomials. All NLControl functions described above work with discrete-time systems as well. One only has to replace the keyword `TimeDerivative` by the word `Shift` in the control object to indicate the different time-domain and in the description of Ore ring use different commutation rule. Non-commutative polynomial approach allows to address a multitude of other problems, not discussed in this paper. Our package development plans in near future include the following tasks. First, one may solve the system linearization problem up to the i/o injection easily using the concept of adjoint polynomials. Second, the degree of the Dieudonné determinant of the control object to indicate the different time-domain and in the description of Ore ring use different commutation rule. Non-commutative polynomial approach allows to address a multitude of other problems, not discussed in this paper. Our package development plans in near future include the following tasks. First, one may solve the system linearization problem up to the i/o injection easily using the concept of adjoint polynomials. Second, the degree of the Dieudonné determinant of the polynomial matrix $P(Z)$ has been used to find the order of the realization by Kotta et al. [2008]. Third, polynomial approach allows to transform the set of i/o equations into different canonical forms being the extensions of Hermite, Popov and Guiderzi forms in the linear case. And fourth, transfer matrix can be represented as a product of two terms, one of the terms being polynomial matrix. The possibility to transform the latter polynomial matrix into Smith-Jacobson form is intended to add to the package.

REFERENCES


