Proceedings of the Estonian Academy of Sciences,
2009, **58**, 4, 224–240
doi: 10.3176/proc.2009.4.04
Available online at www.eap.ee/proceedings

COMPUTER
SCIENCE

WebMathematica-based tools for discrete-time nonlinear control systems

Maris Tõnso*, Heli Rennik, and Ülle Kotta

Institute of Cybernetics, Tallinn University of Technology, Akadeemia tee 21, 12818 Tallinn, Estonia

Received 11 December 2008, revised 22 May 2009, accepted 25 May 2009

Abstract. The package NLControl, developed in the Institute of Cybernetics at Tallinn University of Technology within the Mathematica environment, has been made partially available over the internet using webMathematica tools. The package consists of functions that assist the solution of different modelling, analysis, and synthesis problems for nonlinear control systems, described either by state or by input-output equations. This paper focuses on describing the webMathematica-based tools for discrete-time nonlinear control systems.

Key words: nonlinear control systems, webMathematica, symbolic computations, algebraic framework.

1. INTRODUCTION

The most popular approach in nonlinear control theory is based on differential geometry (Nijmeijer and van der Schaft, 1990; Isidori, 1995). However, by the end of the 1980s its limits became known, meaning that geometric tools were not able to provide the most natural framework when studying problems like (dynamic) feedback linearization, realization, and inversion. The introduction of differential algebraic methods (Fliess, 1990) offered additional tools for investigating these and many other problems. Today, the algebraic point of view has gained popularity and the related approach based on the vector spaces of differential one-forms over suitable fields of nonlinear functions has been introduced (Conte et al., 2007). The tools based on differential one-forms and the related methods based on the theory of the skew polynomial rings are complementary to the differential geometric methods, but what is more important, these tools are characterized by their inherent simplicity and strong similarity to their linear counterparts. The latter makes these tools a better choice in teaching engineering courses in nonlinear control and in practical applications.

Since the solutions of nonlinear control problems require a huge amount of symbolic computations, additional assistance is provided by the nonlinear control system software package NLControl (Kotta and Tõnso, 1999; Kotta and Tõnso, 2003), developed in the Institute of Cybernetics at Tallinn University of Technology. The package is based on algebraic methods of differential one-forms and skew polynomials, and is developed within the (symbolic) software system Mathematica. This package provides basic tools for modelling, analysis, and synthesis both for discrete- and continuous-time nonlinear systems. The reason for developing our own package is that nonlinear control systems, unlike their linear counterparts, practically miss the support of professional software products. For example, both the Matlab Control System Toolbox and Mathematica Control System Professional Suite are applicable only to linear systems. Some custom-made packages based on symbolic computations for nonlinear control systems have been developed but their

* Corresponding author, maris@cc.ioc.ee

distribution is extremely limited and none of them implements the methods based on algebraic tools (see, for example, de Jager, 1995; Rothfuß and Zeitz, 1997; Rodrigues-Millan, 2001; Kaddouri et al., 2006).

The functions from the package NLControl cannot be used outside the Mathematica environment. The main purpose of this paper is to introduce and describe a webMathematica-based application developed by us, which allows the most important functions from NLControl to be made available via the world-wide-web, in such a way that no other software except for an internet browser needs to be installed in a computer to use these tools. This allows these tools to be applied in graduate courses as well as making them available to a wider control community and to engineers. The other web-based tool for a nonlinear control system is described by Ondera and Huba (2006). The authors are not aware of any other web implementations related to symbolic methods for nonlinear control systems. However, there exist numerous applications of webMathematica in control education (see, for example, Kujan et al., 2005), but practically all these tools are dedicated to linear systems. The second purpose of the paper is to compare our tools with those described by Ondera and Huba (2006). Because of space limitations, in this paper we focus only on the discrete-time case. Differences between continuous- and discrete-time cases are only briefly commented upon at the end of the paper.

The paper is organized as follows. Section 2 gives a short overview of the algebraic approach based on differential forms. Section 3 gives an overview of those aspects of webMathematica necessary for our application, including technical details about webMathematica server technology. Section 4 describes functions, implemented in our webMathematica website together with numerous examples. Section 5 presents a brief description of the NLControl package and the structure of the website. Additionally, in Section 5, we make a comparison with the other web-based tools for nonlinear control systems and comment briefly upon the specifics as well as differences of the continuous-time case from the discrete-time case. The next section concludes the paper and after that we have an Appendix, containing source code examples with explanations.

The developed webpage is available at http://webmathematica.cc.ioc.ee/webmathematica/NLControl/

2. ALGEBRAIC TOOLS BASED ON DIFFERENTIAL FORMS

Nonlinear control systems can be, in general, described in many different ways. In this paper we consider three of them: state equations, input-output (i/o) equations, and polynomial representation. First, the system can be described by the state equations

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

where $u \in U \subset \mathbb{R}^m$ is the input, $y \in Y \subset \mathbb{R}^p$ is the output, $x \in X$, an open subset of $\mathbb{R}^n$, is the state, $f : X \times U \to X$ and $h : X \to Y$ are real analytic functions. Alternatively, the control system may be described by the set of higher-order i/o difference equations relating the inputs $u_j, j = 1 \ldots, m$, the outputs $y_i, i = 1, \ldots, p$, and a finite number of their forward time shifts,

$$\begin{gathered} y_i(t+n_i) = \varphi_i(y_s(t), \ldots, y_s(t+n_{is}-1), u_k(t), \ldots, u_k(t+s_{ik}), \\ s = 1, \ldots, p, k = 1, \ldots, m), \quad i = 1, \ldots, p, \end{gathered} \tag{2}$$

where $\varphi = [\varphi_1, ..., \varphi_p]^T$ is a real analytic function, s_{ik}, n_{is}, and n_i are the integer-valued structural parameters of the system, satisfying the conditions $s_{ik} < n_i$, $n_{is} < \min(n_i, n_s)$, and $n_1 + ... + n_p = n$. To use the algebraic tools based on differential forms, as well as the Mathematica functions in the package NLControl, the submersivity assumption has to be satisfied for discrete-time systems; see Assumptions 1 and 2 below, for systems described by equations (1) and (2), respectively.

Assumption 1: System (1) is submersive if the function f satisfies generically (i.e. everywhere except on a set of measure zero) the condition

$$\operatorname{rank}\frac{\partial f(x(t),u(t))}{\partial(x(t),u(t))}=n.$$

Assumption 2: System (2) is submersive if the function φ satisfies generically the condition

$$\operatorname{rank}\frac{\partial \varphi(\cdot)}{\partial(y(t),u(t))}=p.$$

Below we will give a short overview of the algebraic approach based on differential one-forms. We will use the notations from Aranda-Bricaire et al. (1996). One can associate with system (1) the field $\mathscr{K}$ of meromorphic functions in a finite number of independent system variables $\{x(0),u(t),t\geq 0\}$. The forward-shift operator $\delta:\mathscr{K}\to\mathscr{K}$ is defined by $\delta\varphi(x(t),u(t))=\varphi(f(x(t),u(t)),u(t+1))$. Under Assumption 1 the pair $(\mathscr{K},\delta)$ is a difference field (Aranda-Bricaire et al., 1996), and up to an isomorphism, there exists a unique difference field $(\mathscr{K}^*,\delta^*)$, called the inversive closure of $(\mathscr{K},\delta)$, such that $\mathscr{K}\subset\mathscr{K}^*$, $\delta^*:\mathscr{K}^*\to\mathscr{K}^*$ is an automorphism and the restriction of δ^* to $\mathscr{K}$ equals δ. Aranda-Bricaire et al. (1996) give an explicit construction of $(\mathscr{K}^*,\delta^*)$. In what follows we use the same symbol for $\mathscr{K}$ and $\mathscr{K}^*$. Sometimes the abridged notation $\varphi^+(\cdot)=\delta\varphi(\cdot)$ for $\varphi\in\mathscr{K}$ is used.

Over the field $\mathscr{K}$ one can define a vector space $\mathscr{E}:=\operatorname{span}_{\mathscr{K}}\{\mathrm{d}\varphi\mid\varphi\in\mathscr{K}\}$ spanned by the differentials of the elements of $\mathscr{K}$. The elements of $\mathscr{E}$ are called differential one-forms. The forward-shift operator $\delta:\mathscr{K}\to\mathscr{K}$ induces a forward-shift operator $\delta:\mathscr{E}\to\mathscr{E}$ by

$$\textstyle\sum_i a_i\mathrm{d}\varphi_i\to\sum_i a_i^+\mathrm{d}(\delta\varphi_i),a_i,\varphi_i\in\mathscr{K}.$$

A one-form $\omega\in\mathscr{E}$ is called exact if $\mathrm{d}\omega=0$ and closed if $\mathrm{d}\omega\wedge\omega=0$, where $\wedge$ denotes the wedge product. The subspace of one-forms in $\mathscr{E}$ is called completely integrable if it admits the basis which consists only of closed one-forms. The relative degree r of a one-form ω in $\mathscr{X}:=\operatorname{span}_{\mathscr{K}}\{\mathrm{d}x\}$ is defined by $r=\min\{k\in\mathbb{N}\mid\omega(k)\not\subset\mathscr{X}\}$. The relative degree is the number of times one has to apply the forward-shift operator to make the one-form explicitly dependent on the input.

The difference field $\mathscr{K}$ and the shift operator δ induce a ring of left polynomials in the shift operator δ, denoted by $\mathscr{K}[\delta]$ (Kotta and Tõnso, 2004). A left polynomial $p(\delta)\in\mathscr{K}[\delta]$ is written as

$$p(\delta)=a_k\delta^k+a_{k-1}\delta^{k-1}+\ldots+a_1\delta+a_0,$$

where $a_i\in\mathscr{K}$, for $0\leq i\leq k$. Each polynomial $p(\delta)\in\mathscr{K}[\delta]$ is a mapping of $\mathscr{E}$ into itself. An element $a\in\mathscr{K}$ does not commute with the shift operator δ, i.e. $a\cdot\delta\neq\delta\cdot a$. Since the multiplication of δ and an element $a\in\mathscr{K}$ is not commutative and can be defined by the following rule

$$\delta\cdot a=a^+\delta, \tag{3}$$

the ring $\mathscr{K}[\delta]$ thus defined is a non-commutative skew polynomial ring and is proved to be a (left) Ore ring. That is, the polynomials from $\mathscr{K}[\delta]$ satisfy the left Ore condition: for all non-zero $a,b\in\mathscr{K}[\delta]$ there exist non-zero $\alpha,\beta\in\mathscr{K}[\delta]$ such that $\alpha b=\beta a$.

Input-output equations (2) may be alternatively represented as (Kotta and Tõnso, 2004)

$$P(\delta)\mathrm{d}y(t)=Q(\delta)\mathrm{d}u(t), \tag{4}$$

where $P(\delta)$ and $Q(\delta)$ are $p\times p$ and $p\times m$-dimensional matrices, respectively, whose elements $p_{ij},q_{ij}\in\mathscr{K}[\delta]$:

$$p_{is}(\delta)=\delta^{n_i}-\sum\frac{\partial\varphi_i}{\partial y_s(t+j)}\delta^j,q_{ik}(\delta)=\sum\frac{\partial\varphi_i}{\partial u_k(t+r)}\delta^r$$

and $\mathrm{d}y(t)=[\mathrm{d}y_1(t),\ldots,\mathrm{d}y_p(t)]^T,\mathrm{d}u(t)=[\mathrm{d}u_1(t),\ldots,\mathrm{d}u_m(t)]^T$. Equation (4) is obtained from (2) by applying the differential operation to it and using the notations $\mathrm{d}y_s(t+j)=\delta^j\mathrm{d}y_s(t)$, $\mathrm{d}u_k(t+r)=\delta^r\mathrm{d}u_k(t)$.

3. WEBMATHEMATICA

In this section we will give a short overview of webMathematica technology (see Wolfram Research, Inc., webMathematica documentation; http://documents.wolfram.com/webmathematica) and features related to our website. In the webMathematica server the Mathematica kernel is running, which is taking requests inserted by users from the webpage, calculating the results and sending them back to the webpage.

Our website (or user interface) uses standard web graphical user interface elements, such as text fields and check boxes. WebMathematica allows a site to deliver HTML pages to which Mathematica commands have been added, and uses the request/response standard, followed by web servers. In our website request is entered into text fields and response can be in HTML, image or Mathematica notebook form. The request/response process is the following:

1. The browser sends a request to the webMathematica server. The request includes variables and their symbolic values entered from the webpage.
2. The webMathematica's kernel manager acquires the Mathematica kernel. Variables and symbolic values are sent to this kernel.
3. The Mathematica kernel is initialized with input (request) parameters, it carries out the calculations, and returns the result to the server.
4. The response is sent to the browser.
5. The webMathematica server returns the result to the browser.

Requests are sent to the server with webMathematica webpages that are based on two standard Java technologies: Java Servlet and JavaServer Pages (JSP). Servlets are special Java programs that run in a Java-enabled web server, which is typically called a 'servlet container'. JavaServer Pages use a special library of tags that work with Mathematica. This library of tags is called the MSP Taglib. In our site we use also JavaScript and Java. We use JavaScript for opening and closing windows and communicating between windows and Java for calculating random inputs to generate examples.

There are many different combinations of hardware and operating systems that support webMathematica components. Before one starts to install webMathematica, one has to install Java and a servlet container. We are using the Linux operating system and Tomcat as a web container.

4. IMPLEMENTED FUNCTIONS

To date, we have implemented 13 different functions for discrete-time nonlinear control systems from the NLControl package into the webMathematica website. There are seven pages for systems described by state equations and six pages for systems described by i/o equations. Most of the chosen functions are based on subspaces $\mathscr{H}_k$ (Aranda-Bricaire et al., 1996; Kotta et al., 2001) and solve different modelling, analysis, and synthesis problems.

Note that the tools of NLControl are not designed for approximate calculations. Therefore, all real (floating-point) numbers are transformed into rational numbers. Before printing the result, the rational numbers are transformed back to real numbers (keeping the accuracy of the input) for a better overview. The user interface is interconnected with the help system, which provides a detailed explanation describing the applied methods behind the functions together with numerous typical examples.

User input data (i.e. system equations, input, output, and state variables) validation has been implemented in the user interface. The result of the function (together with the applied equations) can be given either in an HTML table, gif picture or pure Mathematica output form.

The implemented functions can be divided into several subgroups. First, there are assistant functions that do not solve any control problems. They either check submersivity assumption (`Submersivity`) or perform the basic operations with Ore polynomials (`OrePolynomials`) or compute the sequence of $\mathscr{H}_k$ subspaces in two different ways (`SequenceHk` and `SequenceHOre`), which is a necessary building block of the majority of other functions. The functions of the second group perform the transformations between different system descriptions (`Realization`, `ClassicStateToIO`, `IOToPolynomials`, `Reduction`

and `NormalForm`). The functions of the third group check the system properties (`Accessibility` and `ObservabilityFiltration`). Finally, the last group consists by now of only one function `FeedbackLinearization`.

4.1. Assistant functions

Some of the assistant functions have been made available on the website to allow deeper study of the control system. Access to assistant functions may be useful if the primary function, supposed to solve a control problem, fails for some reason. Assistant functions allow one to inquire the reasons for failure and sometimes even to solve the problem.

Submersivity. The function `Submersivity` has to return `True` for the other functions to be applicable. The function `Submersivity` returns `True` if equation (1) satisfies Assumption 1. If the result is `False`, then the other Mathematica functions cannot be used, since they may yield wrong results. The typical problems one may encounter in case of non-submersive systems are discussed in Aranda-Bricaire and Kotta (2004). However, note that the submersivity assumption is not restrictive, since it is always satisfied for accessible (controllable) systems.

For checking the `Submersivity` assumption the system has to be given in the state space form with the list of state and input variables. At the moment, there does not exist any submersivity assumption test for i/o systems of the form (2).

Example 1. Consider the bioreactor model, where cells are being grown through the consumption of a substrate (Kazantzis and Kravaris, 2001):

$$\begin{aligned} x_1(t+1) &= x_1(t)+\left[\frac{x_1(t)x_2(t)}{x_1(t)+x_2(t)}-0.08x_1(t)\right]T, \\ x_2(t+1) &= x_2(t)+\left[\frac{-x_1(t)x_2(t)}{x_1(t)+x_2(t)}-0.08x_2(t)+0.008\right]T. \end{aligned}$$

`Submersivity` returns `True`, meaning that the system is submersive.

Ore polynomials. In order to implement functions based on the polynomial methods, for example `Reduction` and `SequenceHOre`, one needs basic tools for working with Ore polynomials (Ore, 1933). In Mathematica, unlike Maple, there are no built-in packages dedicated to Ore polynomials. An important point to emphasize is that the basic operations with Ore polynomials have to be performed modulo system equations (2). Whenever the expression $y_i(t+n_i)$ appears in computations, it should always be replaced by $\varphi_i(\cdot)$ from equations (2). The higher-order time-shifts $y_i(t+\alpha_i), \alpha_i > n_i$ require repeated replacements, for example $y_i(t+n_i+1)$ should be at first substituted by $\delta\varphi_i(\cdot)$ and then each $y_i(t+n_i)$ in the result given by $\varphi_i(\cdot)$. Variables at negative time-instances should be treated in a similar manner, considering that not all of them are independent variables. In order to find expressions for substituting dependent variables, at first one has to choose p variables from the list $\{u_j(t), y_i(t), \ \ j=1,\ldots,m\}$, $i=1,\ldots,p$ such that Assumption 2 is satisfied for the chosen variables. Denote the selected variables by $z_i(t)$, $i=1,\ldots,p$ and solve system (2) for $z_i(t)$. After applying the backward-shift operator to the solution the required number of times, it allows replacing variables $z_i(t)$, $t<0$.

The function `Ore polynomials` operates with polynomials from the Ore ring and completes left and right division, finds left and right greatest common divisor, left and right common multiple, etc.

Example 2. Consider the polynomials $A=\delta^2+u(t)$ and $B=y(t+1)\delta+3$. Then the operation of left division $A=B*Q+R$ returns the result:

$$\begin{aligned} Q &= \frac{1}{y(t)}\delta-\frac{3}{y(t-1)y(t)}, \\ R &= \left(u(t)+\frac{9}{y(t-1)y(t)}\right). \end{aligned}$$

SequenceHk. The subspace $\mathscr{H}_k$ (for $k \geq 0$) contains the one-forms with a relative degree equal to at least k and is defined by

$$\begin{aligned} \mathscr{H}_0 &= \text{span}_{\mathscr{K}}\{\mathrm{d}x, \mathrm{d}u\}, \\ \mathscr{H}_k &= \{\omega \in \mathscr{H}_{k-1} \mid \omega^+ \in \mathscr{H}_{k-1}\},\ k \geq 1. \end{aligned} \tag{5}$$

Computing the sequence of subspaces $\mathscr{H}_k$ is necessary for several purposes. First, we need this sequence for checking realizability property and for finding the classical state space realization of the i/o equation (`Realization`). Second, it is also necessary for checking if the system is accessible or not and in case it is not, to find the non-accessible subspace and decompose the system into accessible and non-accessible subsystems (`Accessibility`). Moreover, this sequence is necessary to check if the system is static state feedback linearizable and for finding the state coordinate transformation (`FeedbackLinearization`). Finally, this sequence is also necessary to check if the i/o equations are irreducible or not and if not, to find the reduced model (`Reduction`), and to transform the state equations into the so-called normal form (`NormalForm`).

The function `SequenceHk` computes the first N elements in the sequence of subspaces $\mathscr{H}_k$, associated either to state equations (1) or to i/o equations (2), where N is a positive integer, specified by the user.

Example 3. This example demonstrates the advantage of the polynomial method for computing the sequence $\mathscr{H}_k$:

$$y(t+3) = y(t) + \sqrt{u(t+2)y(t+1)}. \tag{6}$$

The function `SequenceHk` returns

$$\begin{aligned} \mathscr{H}_1 &= \text{span}_{\mathscr{K}}[\mathrm{d}y(t), \mathrm{d}y(t+1), \mathrm{d}y(t+2), \mathrm{d}u(t), \mathrm{d}u(t+1), \mathrm{d}u(t+2)], \\ \mathscr{H}_2 &= \text{span}_{\mathscr{K}}[\mathrm{d}y(t), \mathrm{d}y(t+1), \mathrm{d}y(t+2), \mathrm{d}u(t), \mathrm{d}u(t+1)], \\ \mathscr{H}_3 &= \text{span}_{\mathscr{K}}[\mathrm{d}y(t), \mathrm{d}y(t+1), \mathrm{d}u(t), 2\sqrt{u(t+1)}\mathrm{d}y(t+2) - \sqrt{y(t)}\mathrm{d}u(t+1)], \\ \mathscr{H}_4 &= \{0\}. \end{aligned}$$

It is possible to show, proceeding from the definition (5), that in case of the single-input single-output (SISO) system the number of basis one-forms decreases by 1 at every step. Since $\mathscr{H}_3$ in the above example contains four basis one-forms, $\mathscr{H}_4$ should contain three basis one-forms, instead of being $\{0\}$, as returned by the function `SequenceHk`. The error occurs since the algorithm used to compute the sequence $\mathscr{H}_k$ requires the solution of a complicated system of nonlinear equations, containing square roots in the above case. Mathematica fails to solve the system, thus correct basis one-forms for $\mathscr{H}_4$ cannot be found. Within NLControl the function `SequenceHk` in this case displays an error message. Unfortunately, the webMathematica version of this function does not display the error message yet.

In such cases an alternative algorithm, which does not require solving a system of equations, may be useful. The alternative algorithm is based on Ore polynomials and at present is available only for SISO systems.

SequenceHOre. In the SISO case, if the system is described by equation (4), i.e. by

$$p(\delta)\mathrm{d}y(t) = q(\delta)\mathrm{d}u(t), \tag{7}$$

the subspaces $\mathscr{H}_k$ can be computed directly from polynomials $p(\delta)$ and $q(\delta)$ in (7). The polynomial method allows computing only the first $s+2$ elements of the sequence $\mathscr{H}_k$, which is exactly the number of elements the application of the function `Realization` requires. However, at the moment, the default method used by the function `Realization` is `SequenceHk`. The reason for this is that `SequenceHk` is applicable also in the multi-input multi-output (MIMO) case, unlike `SequenceHOre`. Our further goal is to make the use of the `SequenceHOre` method also available within `Realization`. This method is more direct and in the discrete-time case, also noticeably faster.

For the i/o model (7) the subspaces $\mathscr{H}_k$ for $k = 2, ..., s+2$ can be calculated as (Kotta and Tõnso, 2008)

$$\mathscr{H}_k = \mathrm{span}_{\mathscr{K}}\{\mathrm{d}y(t), ..., \mathrm{d}y(t+n-k+1), \mathrm{d}u(t), ..., \mathrm{d}u(t+s-k+1), \omega_l\},$$

where $l = 1, ..., k-2$ and

$$\omega_l = \sum_{i=n-k+1}^{n-l} \delta^{-l} p_{l+i} \delta^i \mathrm{d}y(t) + \sum_{j=s-k+2}^{s-l} \delta^{-l} q_{l+j} \delta^j \mathrm{d}u(t).$$

The Mathematica block sent to the server looks almost the same as in the case of `SequenceHk`.

Example 4. In the case of the previous example with the polynomial method the function `SequenceHOre` returns

$$\begin{aligned}
\mathscr{H}_1 &= \mathrm{span}_{\mathscr{K}}[\mathrm{d}y(t), \mathrm{d}y(t+1), \mathrm{d}y(t+2), \mathrm{d}u(t), \mathrm{d}u(t+1), \mathrm{d}u(t+2)],\\
\mathscr{H}_2 &= \mathrm{span}_{\mathscr{K}}[\mathrm{d}y(t), \mathrm{d}y(t+1), \mathrm{d}y(t+2), \mathrm{d}u(t), \mathrm{d}u(t+1)],\\
\mathscr{H}_3 &= \mathrm{span}_{\mathscr{K}}\left[\mathrm{d}y(t), \mathrm{d}y(t+1), \mathrm{d}u(t), \mathrm{d}y(t+2) - \frac{y(t)}{2\sqrt{u(t+1)y(t)}}\mathrm{d}u(t+1)\right],\\
\mathscr{H}_4 &= \mathrm{span}_{\mathscr{K}}\left[\mathrm{d}y(t), \mathrm{d}y(t+1) - \frac{-\sqrt{u(t+1)y(t)} + y(t+2)}{2\sqrt{u(t)(-\sqrt{u(t+1)y(t)} + y(t+2))}}\mathrm{d}u(t),\right.\\
&\qquad \left.\mathrm{d}y(t+2) - \frac{y(t)}{2\sqrt{u(t+1)y(t)}}\mathrm{d}u(t+1)\right].
\end{aligned}$$

4.2. Transformations between different system descriptions

In this subsection the functions allowing transformation of one system description into another are described. First, `ClassicStateToIO` finds i/o equations (2) from state equations (1). Second, the function `Realization` checks if i/o equations (2) can be transformed into the state space form, and finds the state equations, if possible. The function `IOToPolynomials` calculates the matrices $P(\delta)$ and $Q(\delta)$ in (4), given i/o equations (2). The function `Reduction` checks if the i/o equations (2) can be reduced and finds the minimal equivalent system description. Finally, the function `NormalForm` transforms the state equations into the so-called normal form, which is a key factor in applying the inversion-based control methods.

Reduction. The function `Reduction` determines whether the system described by i/o equations (2) is irreducible or not, and if not, finds the reduced set of i/o equations. A nonlinear control system described by equations (2) is irreducible iff the greatest common left divisor $G_L(\delta)$ of polynomial matrices $P(\delta)$ and $Q(\delta)$ in (4) is a unimodular matrix (i.e. a polynomial matrix with a polynomial inverse) (Kotta and Tõnso, 2004). In case of the non-unimodular $G_L(\delta)$, equation (4) may be rewritten as

$$G_L(\delta)[\tilde{P}(\delta)\mathrm{d}y(t) - \tilde{Q}(\delta)\mathrm{d}u(t)] = 0$$

and the reduced system equations can be found by integrating the one-forms in the brackets, perhaps after multiplying by the integrating factors.

Example 5. Consider the system

$$\begin{aligned}
y_1(t+2) &= y_1(t+1) - u_1(t)y_1(t) + u_1(t+1)y_1(t+1),\\
y_2(t+3) &= y_2(t+2)u_2(t+2) + y_2(t+1) - u_1(t+1)y_1(t+1) - 3y_1(t+1)\\
&\quad + 3u_1(t)y_1(t) + y_1(t) - u_2(t)y_2(t).
\end{aligned}$$

The function `Reduction` returns the following reduced equations:

$$\begin{aligned} y_1(t+1) &= u_1(t)y_1(t), \\ y_2(t+1) &= y_2(t)u_2(t) - y_1(t). \end{aligned}$$

Example 6. Consider the system

$$y(t+2) = e^{\sin(u(t)y(t)+u(t+1)\log(y(t+1)))}.$$

After 30 seconds the time-out notice is given.

Realization. The realization problem is to construct state equations (1) of the order $n = n_1 + ... + n_p$ from the set of i/o equations (2), if possible. Note that unlike in the linear case, the state-space realization does not exist for every nonlinear i/o model (2). The necessary and sufficient realizability conditions require that the subspaces $\mathscr{H}_k$, associated with i/o equation (2), for $1 \le k \le s+2$ are completely integrable. The state coordinates can be found by integrating the basis one-forms of $\mathscr{H}_{s+2}$.

The function `Realization` determines whether i/o equations (2) can be transformed into the state-space form and in case of the positive answer finds the state equations.

Example 7. The model of a grain drying process is described by the i/o equation (Kotta and Nurges, 1985)

$$\begin{aligned} y(t+3) =& 1.6389y(t+2) - 0.4397y(t+1) - 0.1803y(t) \\ & - 0.0082y(t+2)u(t+2) - 0.0042y(t+1)u(t+1) \\ & - 0.0074y(t)u(t) + 0.0021u(t) + 0.0019u(t+2) - 0.0041u(t+1). \end{aligned} \tag{8}$$

The function `Realization` returns

$$\begin{aligned} x_1(t+1) &= u(t)(0.0019 - 0.0082x_1(t)) + x_2(t), \\ x_2(t+1) &= u(t)(-0.001 - 0.0176x_1(t)) + x_3(t), \\ x_3(t+1) &= u(t)(-0.004 - 0.0327x_1(t)) - 0.1803x_1(t) - 0.4397x_2(t) + 1.6389x_3(t), \\ y(t) &= x_1(t). \end{aligned}$$

IO to polynomials. `IOToPolynomials` computes the $p \times p$ and $p \times m$-dimensional matrices $P(\delta)$ and $Q(\delta)$ in (4), given i/o equations (2).

Example 8. Consider an i/o equation

$$y(t+2) = \sin(y(t)) + \cos(u(t)) + u(t+1).$$

The function `IOToPolynomials` returns

$$P(\delta) = (\delta^2 - \cos(y(t))), Q(\delta) = (\delta - \sin(u(t))).$$

Normal form. Recall that $\mathscr{H}_2$ is the subspace of the one-forms whose relative degrees are greater than or equal to two, which implies that their time-shifts do not depend on control. Integrability of $\mathscr{H}_2$ is a necessary and sufficient condition for transforming equations (1) by state coordinate transformation into the so-called normal form (Kotta, 2000):

$$\begin{aligned} \zeta_{i1}(k+1) &= \zeta_{i2}(k), \\ &\vdots \\ \zeta_{i,ri-1}(k+1) &= \zeta_{i,ri}(k), \\ \zeta_{i,ri}(k+1) &= \psi_i(\zeta(k), \eta(k), u(k)), \quad i = 1, ..., p, \\ \eta(k+1) &= \gamma(\zeta(k), \eta(k), u(k)), \\ \varphi_i(k) &= \zeta_{i1}(k), \end{aligned}$$

where the dynamics of unobservable states η does not depend on control. Note that the state coordinates can be found by integrating the bases of one-forms of $\mathscr{H}_2$.

Example 9. Consider the discrete-time nonlinear control system

$$\begin{aligned} x_1(t+1) &= x_3(t), \\ x_2(t+1) &= u_1(t)u_2(t)+x_2(t), \\ x_3(t+1) &= x_1(t)x_4(t), \\ x_4(t+1) &= u_1(t)+x_5(t), \\ x_5(t+1) &= x_3(t)x_5(t), \\ y_1(t) &= x_1(t), \\ y_2(t) &= x_2(t). \end{aligned} \tag{9}$$

The function `NormalForm` returns

$$\begin{aligned} z_1(t+1) &= z_2(t), \\ z_2(t+1) &= z_3(t), \\ z_3(t+1) &= z_2(t)(u_1(t)+z_5(t)), \\ z_4(t+1) &= u_1(t)u_2(t)+z_4(t), \\ z_5(t+1) &= x_3(t)x_5(t), \\ y_1(t) &= z_1(t), \\ y_2(t) &= z_4(t), \end{aligned}$$

where $z_1(t)=x_1(t)$, $z_2(t)=x_3(t)$, $z_3(t)=x_1(t)x_4(t)$, $z_4(t)=x_2(t)$, and $z_5(t)=x_5(t)$.

Classic state to IO. The function `ClassicStateToIO` finds i/o equations (2) from state equations (1).

Example 10. Consider the state equations

$$\begin{aligned} x_1(t+1) &= x_1(t)u(t)+x_3(t), \\ x_2(t+1) &= x_3(t)-x_2(t), \\ x_3(t+1) &= x_3(t)+x_1(t)u(t)-x_2(t), \\ y(t) &= x_1(t). \end{aligned} \tag{10}$$

The function `ClassicStateToIO` returns

$$y(t+3)=u(t)y(t)+u(t+1)y(t+1)+u(t+2)y(t+2).$$

4.3. Checking the system properties

In this subsection the functions `Accessibility` and `ObservabilityFiltration` are described that allow checking the accessibility (controllability) and observability properties of the system, respectively.

Accessibility. Accessibility is the structural property of the nonlinear system which in the linear case reduces to the controllability property. System (1) is said to be accessible if there does not exist any non-zero autonomous variable for (1) in $\mathscr{K}$. Note that the autonomous variable of the system is a variable, not influenced by control, and therefore, cannot be changed by the controller. The necessary and sufficient condition for accessibility is $\mathscr{H}_\infty=\{0\}$ (Aranda-Bricaire et al., 1996).

`Accessibility` returns `True` if the nonlinear system is accessible and `False` otherwise. If the system is not accessible, it can be decomposed into accessible and non-accessible subsystems. Note that $\mathscr{H}_\infty$ is a non-accessible subspace, and because of its complete integrability, it has an integrable basis $\mathscr{H}_\infty = \text{span}_{\mathscr{K}}\{\mathrm{d}\zeta_1, ..., \mathrm{d}\zeta_r\}$ (Aranda-Bricaire et al., 1996). Since $\mathscr{H}_\infty$ is invariant under applying the forward shift operator,

$$\begin{aligned} \zeta_1(t+1) &= f_1(\zeta_1(t), ..., \zeta_r(t)), \\ &\vdots \\ \zeta_r(t+1) &= f_r(\zeta_1(t), ..., \zeta_r(t)). \end{aligned}$$

The accessible subspace $\mathscr{X}_a := \mathscr{X}/\mathscr{H}_\infty$ such that $\mathscr{X}_a \oplus \mathscr{H}_\infty = \mathscr{X}$ has also an integrable basis $\text{span}_{\mathscr{K}}\{\mathrm{d}\zeta_{r+1}, ..., \mathrm{d}\zeta_n\}$. Therefore, we have

$$\begin{aligned} \zeta_{r+1}(t+1) &= f_{r+1}(\zeta(t), u(t)), \\ &\vdots \\ \zeta_n(t+1) &= f_n(\zeta(t), u(t)). \end{aligned}$$

For applying the function `Accessibility` the system has to be given in the state space form.

Example 11. Consider a system

$$\begin{aligned} x_1(t+1) &= u(t)^2 + 2x_1(t) - 2x_2(t) + x_2(t)u(t), \\ x_2(t+1) &= u(t)^2 + x_1(t) - x_2(t), \\ x_3(t+1) &= u(t)x_2(t) + x_3(t). \end{aligned} \tag{11}$$

Accessibility decomposition divides system (11) into a non-accessible subsystem

$$z_1(t+1) = z_1(t)$$

and an accessible subsystem

$$\begin{aligned} z_2(t+1) &= u(t)^2 + 2z_2(t) - 2z_3(t) + u(t)z_3(t), \\ z_3(t+1) &= u(t)^2 + z_2(t) - z_3(t), \end{aligned}$$

where

$$\begin{aligned} z_1(t) &= x_1(t) - x_2(t) - x_3(t), \\ z_2(t) &= x_1(t), \\ z_3(t) &= x_2(t). \end{aligned}$$

Observability filtration. (Kotta, 2005) System (1) is said to be observable if

$$\text{rank}_{\mathscr{K}} \frac{\partial H_{n-1}}{\partial x} = n, \tag{12}$$

where $H_{n-1} = \left(h(x), \delta h(x), \ldots, \delta^{n-1}h(x)\right)^T$. Define the difference vector spaces $\mathscr{Y}^k$, $\mathscr{Y}$, and $\mathscr{U}$ for system (1) as follows:

$$\begin{aligned} \mathscr{Y}^k &= \text{span}_{\mathscr{K}}\{\mathrm{d}y(t), 0 \le t \le k\}, \\ \mathscr{Y} &= \text{span}_{\mathscr{K}}\{\mathrm{d}y(t), t \ge 0\}, \\ \mathscr{U} &= \text{span}_{\mathscr{K}}\{\mathrm{d}u(t), t \ge 0\}. \end{aligned}$$

To define the observable subspace, introduce the sequence of subspaces, called the observability filtration

$$0 \subset \mathscr{O}_0 \subset \mathscr{O}_1 \ldots \subset \mathscr{O}_k \subset \ldots, \tag{13}$$

where $\mathscr{O}_k := \mathscr{X} \cap (\mathscr{Y}^k + \mathscr{U})$. The subspace $\mathscr{X} \cap (\mathscr{Y} + \mathscr{U})$ is called the observable space of system (1) and can be computed as the limit $\mathscr{O}_\infty$ of the observability filtration (13), $\mathscr{O}_\infty = \mathscr{X} \cap (\mathscr{Y} + \mathscr{U})$. The following statements are equivalent: (i) the system is observable, (ii) the condition (12) is satisfied, and (iii) $\mathscr{O}_\infty = \mathscr{X}$. In order to decompose system (1) into the observable and unobservable subsystems, $\mathscr{O}_\infty$ has to be integrable. Note that in the continuous-time case $\mathscr{O}_\infty$ is always integrable (Conte et al., 2007), but this is not necessarily true for discrete-time systems. If the observable space $\mathscr{O}_\infty$ is integrable, it admits an exact basis $\{\mathrm{d}\zeta_1, \ldots, \mathrm{d}\zeta_r\}$. Complete the set $\{\mathrm{d}\zeta_1, \ldots, \mathrm{d}\zeta_r\}$ to a basis $\{\mathrm{d}\zeta_1, \ldots, \mathrm{d}\zeta_r, \mathrm{d}\zeta_{r+1}, \ldots, \mathrm{d}\zeta_n\}$ of $\mathscr{X}$. Then, in the coordinates ζ, system (1) reads as

$$\begin{aligned}
\zeta_1(t+1) &= f_1(\zeta_1(t), \ldots, \zeta_r(t), u(t)), \\
&\vdots \\
\zeta_r(t+1) &= f_r(\zeta_1(t), \ldots, \zeta_r(t), u(t)), \\
\zeta_{r+1}(t+1) &= f_{r+1}(\zeta(t), u(t)), \\
&\vdots \\
\zeta_n &= f_n(\zeta(t), u(t)), \\
y(t) &= h(\zeta_1(t), \ldots, \zeta_r(t)).
\end{aligned}$$

Example 12. Consider the nonlinear control system

$$\begin{aligned}
x_1(t+1) &= -u(t)x_1(t) + x_3(t), \\
x_2(t+1) &= x_2(t), \\
x_3(t+1) &= u(t)x_1(t) + x_3(t), \\
y(t) &= x_2(t)x_3(t).
\end{aligned}$$

To check whether the system is observable or not, the observability filtration can be computed:

$$\begin{aligned}
\mathscr{O}_1 &= \operatorname{span}_{\mathscr{K}}[x_3(t)\mathrm{d}x_2(t) + x_2(t)\mathrm{d}x_3(t)], \\
\mathscr{O}_2 &= \mathscr{O}_3 = \operatorname{span}_{\mathscr{K}}[-x_3(t)\mathrm{d}x_1(t) + x_1(t)\mathrm{d}x_3(t), x_2(t)\mathrm{d}x_1(t) + x_1(t)\mathrm{d}x_2(t)].
\end{aligned}$$

Note that $\mathscr{O}_\infty = \mathscr{O}_2 \neq \mathscr{X}$, and therefore, the system is not observable.

4.4. Feedback linearization

System (1) is said to be static state feedback linearizable if there exist a state diffeomorphism

$$\tilde{x}(t) = \Phi(x(t))$$

and a regular static state feedback of the form

$$u(t) = \alpha(x(t), v(t)),$$

with $\operatorname{rank}_{\mathscr{K}}[\partial\alpha(\cdot)/\partial v] = m$, such that in the new coordinates the compensated system equations are in the form

$$\begin{aligned}
\tilde{x}_{i1}(t+1) &= \tilde{x}_{i2}(t), \\
&\vdots \\
\tilde{x}_{ik_i-1}(t+1) &= \tilde{x}_{ik_i}(t), \\
\tilde{x}_{ik_i}(t+1) &= v_i(t), \quad i = 1, \ldots, m.
\end{aligned}$$

For linearizability conditions we need to define an integer k^* for the sequence $\mathscr{H}_k$. There exists an integer $k^* \leq n$ such that, for $0 \leq k \leq k^*$, $\mathscr{H}_{k+1} \subset \mathscr{H}_k$ but $\mathscr{H}_{k+1} \neq \mathscr{H}_k$ and $\mathscr{H}_{k^*+1} = \mathscr{H}_{k^*+2} = \ldots = \mathscr{H}_\infty$. The necessary and sufficient conditions for feedback linearizability are
1. $\mathscr{H}_\infty = \{0\}$,
2. $\mathscr{H}_k$ is completely integrable for $1 \leq k \leq k^*$.

Not every accessible system can be linearized by static state feedback. The function `Linearization` checks whether this is possible and in the case of an affirmative answer finds the state coordinate change, the feedback, and the linear closed-loop equations.

Example 13. Consider the discrete-time nonlinear control system given in Baheti et al. (1980):

$$\begin{aligned} x_1(t+1) &= b_1 u(t) x_1(t) + x_2(t), \\ x_2(t+1) &= (a_2 + (a_1 b_1 + b_2) u(t)) x_1(t) + a_1 x_2(t). \end{aligned}$$

The function `Linearization` returns the feedback linearized equations

$$\begin{aligned} z_1(t+1) &= z_2(t), \\ z_2(t+1) &= v_1(t) \end{aligned}$$

together with the new state coordinates

$$\begin{aligned} z_1(t) &= \frac{a_1 b_1 x_1(t) + b_2 x_1(t) - b_1 x_2(t)}{a_1 b_1 + b_2}, \\ z_2(t) &= \frac{-a_2 b_1 x_1(t) + b_2 x_2(t)}{a_1 b_1 + b_2} \end{aligned}$$

and static state feedback

$$v_1(t) = (a_2 + b_2 u(t)) x_1(t) + (a_1 + b_1 u(t)) z_2(t).$$

5. ADDITIONAL COMMENTS

5.1. About the NLControl package

This section gives a short overview of the NLControl package from the point of view of programming. In order to keep the paper focused, only the part of the package available on the webMathematica site is described here. The NLControl package source code is divided into separate files (see Fig. 1). Using separate files requires division of the code into pieces of reasonable size, which gives a better overview of the package. It also simplifies cooperation between different programmers. When working with Mathematica, only the files which are required for computations will be loaded into memory. For instance, when the user starts the NLControl package and enters the task requiring computing the sequence $\mathscr{H}_k$ for a discrete-time system, only the files Core, Ore polynomials, and Sequences are loaded. When using NLControl via webMathematica, all package files have to be loaded at once due to technical restrictions.

In order to make computations in Mathematica with equations of the form (1), a special object has been created in the NLControl package – `StateSpace[`f`, `Xt`, `Ut`, `t`, `h`, `Yt`, Shift ]`. In this object $f = \{f_1, f_2, \ldots\}$ is a list of state transition functions, $Xt = \{x_1(t), x_2(t), \ldots\}$ is a list of state variables, $Ut = \{u_1(t), u_2(t), \ldots\}$ is a list of input variables, t is time, $h = \{h_1, h_2, \ldots\}$ is a list of output functions, $Yt = \{y_1(t), y_2(t), \ldots\}$ is a list of output variables, and `Shift` indicates that one handles the discrete-time system. Object `IO[`eqs`, `Ut`, `Yt`, `t`, Shift ]` is used for representing i/o equations (2). In above, $eqs = \{y_1(t+n_1) == \varphi_1, y_2(t+n_2) == \varphi_2, \ldots\}$ is a list of equations where φ_i comes from (2); Ut, Yt, and t are as above. As conventional for Mathematica, instead of the lists of input and output variables

a single variable may be entered in case of the SISO system. However, `StateSpace` and `IO` always embrace the single variable with curly brackets. This guarantees that the SISO and MIMO systems have the same data structure and consequently, the same programs can handle them. Most of the assistant functions or low-level functions require separate programs for discrete- and continuous-time systems. The higher level functions (`Linearization`, `Accessibility`, `Observability` and transformations between different system descriptions) can share the same source code for discrete- and continuous-time systems.

Below we comment briefly on the NLControl package files, represented as blocks in Fig. 1.

Core The basic objects like `StateSpace` and `IO` are defined here. The file also contains functions for testing submersivity property and computing forward- and backward-shift operators on $\mathscr{K}$ and $\mathscr{E}$. Quite a large part of the Core file occupies the code for printing objects `StateSpace` and `IO` in the traditional form.

Ore polynomials The object for Ore polynomial `OreP`$[a_k,\ldots,a_0]$ is introduced here. The file also includes all functions to make operations with Ore polynomials.

Web contains additional formatting functions necessary to print the system equations and Ore polynomials on the website in the user-friendly form.

Sequences contains functions for computing the sequence $\mathscr{H}_k$.

Integration contains functions which allow checking integrability and integrating the system of one-forms.

Transformations, **Accessibility/Observability**, and **Linearization** contain the functions described, respectively, in Sections 4.2, 4.3, and 4.4.

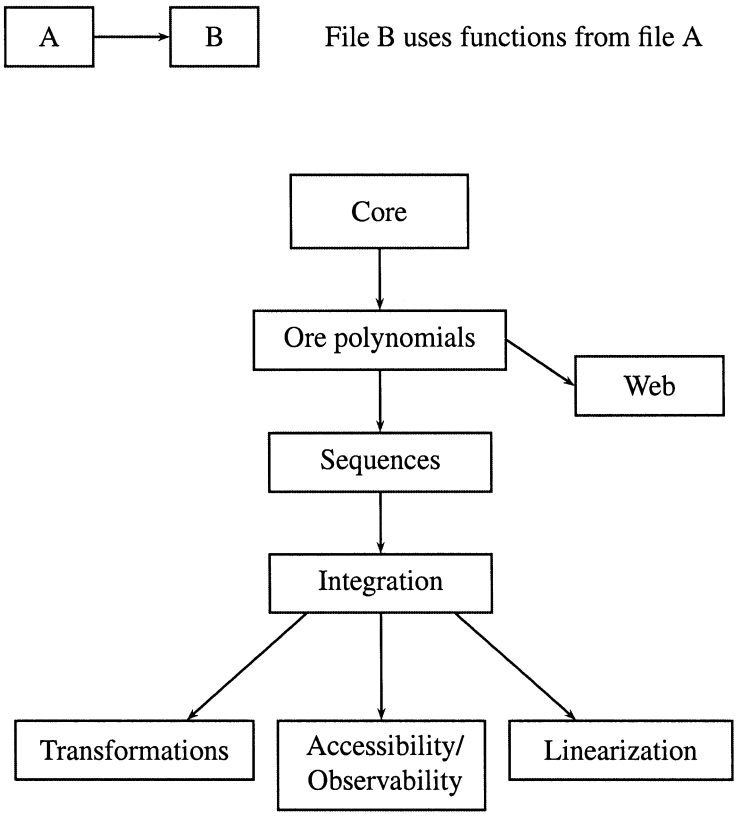

Fig. 1. The NLControl package file structure.

5.2. Structure of the website

The NLControl website is a project under development; we are adding new functions and reorganizing old pages. The structure of the website is very simple. For every function there is one jsp file which allows entry of the input data using HTML forms and the other jsp file which shows the result generated by webMathematica. These two files are similar for all functions, with some modifications. The simplicity of the file structure is also its weakness, since it yields a large number of almost identical files. Moreover, if one has to make some minor changes, one may need to edit all these files. Our future aim is to reorganize the site in a more sophisticated way, such that it would be easier to administer, for example add new functions, update Mathematica code blocks or change the layout.

5.3. Comparison with other web-based tools

Besides the fact that the problems handled are different and the tools described by Ondera and Huba (2006) are dedicated solely to the exact static state feedback linearization problem in the continuous-time case, there are other points to be mentioned. As far as the linearization problem is concerned, the web-tools in Ondera and Huba (2006) allow more than our function `Linearization`, namely the user may additionally submit the desired closed-loop poles of a pole-placement controller and to perform a simulation of the resulting closed-loop system.

There is also a difference in the chosen technology. The web-tools in Ondera and Huba (2006) are based on Matlab and its Symbolic Math Toolbox. This toolbox is a Maple 8 symbolic kernel that was bought from Maplesoft and implemented into Matlab by the MathWorks. The different platform also implies a different internet implementation. In Ondera and Huba (2006) tools are web-accessible via the Matlab Web Server that is based on the CGI technology, whereas webMathematica is Java- and JavaScript-based. Both web-tools relieve users from installing Mathematica or Matlab on their computers and help to make programs available to everyone without seeing the program code.

5.4. Specifics of the continuous-time case

Although in this paper only the functions for the discrete-time case are discussed, the website provides the same functionality also for continuous-time nonlinear systems. The functions corresponding to the continuous-time case use the same naming convention as the discrete-time ones and are distinguished from them by the function argument. The objects containing the control system, that is `StateSpace` and `IO`, have a special argument to indicate the time domain of the system: it is `Shift` for discrete-time systems and `TimeDerivative` for continuous-time systems. However, the web interface user should be careful to select the correct functions from the menu.

The most important difference between discrete- and continuous-time cases is in the submersivity assumption, which needs to be tested in the discrete-time case, but is inherently satisfied in the latter case. The second important difference is in the decomposition of the system into observable and unobservable subspaces. Whereas in the continuous-time case this can always be done, in the discrete-time case one has first to check whether the observable subspace is integrable, which is the necessary condition to perform the decomposition. As for the operations with Ore polynomials, the main difference comes from a different commutation rule, employed in the continuous-time case

$$\mathrm{d}/\mathrm{d}t \cdot a = a \cdot \mathrm{d}/\mathrm{d}t + \dot{a}, \tag{14}$$

compared to that of (3) in the discrete-time case. However, both commutation rules, (14) and (3), yield the Ore polynomial ring.

6. CONCLUSION

The paper describes how the symbolic computation package NLControl, developed within the Mathematica environment, has been made available over the internet using webMathematica programming features. The website has been tested on Microsoft Internet Explorer and Mozilla Firefox web browsers. Our future goal is to implement more NLControl functions into the webMathematica website, and improve the documentation and example library. Especially, we want to include functions that allow output feedback linearization and/or decoupling, construct the transfer function from i/o equations (2) or from state equations (1), find the discrete-time model from the continuous-time system equations, and solve the model-matching problem.

APPENDIX

Below the source code of the Mathematica blocks, which generate the result to the webpage of the function `Realization`, is given. The first block deals with the data entered by the user and the second block performs the actual realization procedure.

```
<msp:evaluate>
  eqs = Null;  eqs = MSPToExpression[$$eqs];
  Ut = Null;  Ut = MSPToExpression[$$Ut];
  Yt = Null;  Yt = MSPToExpression[$$Yt];
  correctdata = eqs=!= Null;
  If[ correctdata,
    If[ Ut === Null, Ut = DetectVariables[eqs, t, "u*"]];
    If[ Yt === Null, Yt = DetectVariables[eqs, t, "y*"]];
    data = IO[ eqs, Ut, Yt, t, Shift];
    timearg = If[ $$argumentt === "yes", False, True];
    StringJoin[
      "Your system is:<br><br>",
       ToString[ iWebForm[ data, $$outputformat,
                           TimeArgument -> timearg]]
    ], (* End StringJoin *)
    (* Else *)
    "Uncorrect input data."
  ] (* End If *)
</msp:evaluate>
```

Variables `$$eqs`, `$$Ut`, and `$$Yt` store the data entered by the user on the input data page into the text fields, `$$outputformat` and `$$argumentt` carry the information about user's formatting preferences. The function `MSPToExpression` turns the string obtained from the text field into the Mathematica expression. If this process fails with the i/o equations, that is if the value of the variable `correctdata` is not `True`, an error message is printed and no more computations are made. However, the failure in the interpretation of the lists of input and output variables does not stop the work of the program; in this case all the variables in the i/o equations starting with `u` and depending on argument `t` (e.g. `u[t]`, `u1[t+1]`, `ua[t+2]`,...) are considered as input variables and analogously those starting with `y` are considered as output variables. The i/o equations are assigned to variable `data` and then printed to the webpage using the special formatting command `iWebForm`. Note that `DetectVariables` and `iWebForm` are defined in the NLControl package and are not Mathematica built-in functions.

```
<msp:evaluate>
  If[ correctdata,
    result = TimeConstrained[ Realization[ data,
                ToExpression["x" <> ToString[#]][t]& ], 28];
    Switch[ result,
      {}, "Classical state space form does not exist for the
                                system.",
      _Realization, "Program is unable to find the classical
                                state space form for the system.",
      $Aborted, "Time limit 30 seconds exceeded.",
      {_StateSpace, _List}, StringJoin[
        "Classical state space equations are:<br><br>",
        iWebForm[ result, $$outputformat, TimeArgument -> timearg]
        ] (* End StringJoin *)
    ] (* End Switch *)
  ] (* End If *)
</msp:evaluate>
```

In the case of complicated systems performing the function `Realization` may often lead to very long calculations, therefore `TimeConstrained` interrupts the computations when the time limit 28 seconds is exceeded. The second argument of the `Realization` function tells Mathematica that the state variables should be denoted as x1, x2, ... The outcome of the realization is assigned to the variable `result`. After that, according to the value of `result`, the corresponding message is printed and in case the result contains the correct state equations, the equations formatted with `iWebForm`, are also included.

REFERENCES

Aranda-Bricaire, E. and Kotta, Ü. A geometric solution to the dynamic disturbance decoupling for discrete-time nonlinear systems. *Kybernetika*, 2004, **40**, 197–206.

Aranda-Bricaire, E., Kotta, Ü., and Moog, C. Linearization of discrete-time systems. *SIAM J. Control Optim.*, 1996, **34**, 1999–2023.

Baheti, R., Mohler, R., and Spang, H. Second-order correlation method for bilinear system identification. *IEEE Trans. Autom. Control*, 1980, **25**, 1141–1146.

Conte, G., Moog, C., and Perdon, A. *Algebraic Methods for Nonlinear Control Systems*. Springer, London, 2007.

de Jager, B. The use of symbolic computations in nonlinear control. Is it viable? *IEEE Trans. Autom. Control*, 1995, **40**, 84–89.

Fliess, M. Automatique en temps discret et algèbre aux différences. *Forum Math.*, 1990, **2**, 213–232.

Isidori, A. *Nonlinear Control Systems (3rd ed.)*. Springer, Berlin, 1995.

Kaddouri, A., Blais, S., Ghribi, M., and Akhrif, O. NLSOFT: An interactive graphical software for designing nonlinear controllers. *Math. Comput. Simulation*, 2006, **71**, 377–384.

Kazantzis, N. and Kravaris, C. Discrete-time nonlinear observer design using functional equations. *Systems Control Lett.*, 2001, **42**, 81–94.

Kotta, Ü. Comments on "On the discrete-time normal form". *IEEE Trans. Autom. Control*, 2000, **45**, 2197.

Kotta, Ü. Decomposition of discrete-time nonlinear control systems. *Proc. Estonian Acad. Sci. Phys. Math.*, 2005, **54**, 154–161.

Kotta, Ü. and Nurges, Ü. Identification of input-output bilinear systems. In *A Bridge Between Control Science and Technology: Proceedings of the Ninth Triennial World Congress of IFAC, Budapest, Hungary, 2–6 July 1984, Vol. 2* (Gertler, J. and Keviczky, L., eds). Pergamon, Oxford, 1985, 723–727 (*IFAC Proceedings* series, 1985, 1).

Kotta, Ü. and Tõnso, M. Transfer equivalence and realization of nonlinear higher order input/output difference equations using Mathematica. *J. Circuits Systems Comput.*, 1999, **9**, 23–25.

Kotta, Ü. and Tõnso, M. Linear algebraic tools for discrete-time nonlinear control systems with Mathematica. In *Nonlinear and Adaptive Control, NCN4 2001* (Zinober, A. and Owens, D., eds), *Lecture Notes in Control and Inform. Sci.*, 2003, **281**, 195–205, Springer, Berlin.

Kotta, Ü. and Tõnso, M. Irreducibility conditions for discrete-time nonlinear multi-input multi-output systems. In *6th IFAC Symposium on Nonlinear Control Systems (NOLCOS)* (Allgöwer, F., ed.). Stuttgart, Germany, 2004, 269–274.

Kotta, Ü. and Tõnso, M. Realization of discrete-time nonlinear input-output equations: polynomial approach. In *7th World Congress on Intelligent Control and Automation, Chongqing, China*. IEEE, 2008, 529–534.

Kotta, Ü., Zinober, A., and Liu, P. Transfer equivalence and realization of nonlinear higher order input-output difference equations. *Automatica*, 2001, **37**, 1771–1778.

Kujan, P., Hromčik, M., and Šebek, M. Web-based Mathematica platform for systems and control education. In *Proceedings of the 13th Mediterranean Conference on Control and Automation, Limassol, Cyprus*. IEEE, Limassol, 2005, 376–381.

Nijmeijer, H. and van der Schaft, A. *Nonlinear Dynamical Control Systems*. Springer, New York, 1990.

Ondera, M. and Huba, M. Web-based tools for exact linearization control design. In *Proceedings of the 14th IEEE Mediterranean Conference on Control and Automation, Ancona, Italy, 28–30 June 2006*. IEEE (CD ROM).

Ore, O. Theory of non-commutative polynomials. *Annals Math.*, 1933, **34**, 480–508.

Rodrigues-Millan, J. Integrated symbolic-graphic-numeric analysis and design in nonlinear control through notebooks in Mathematica. *Lecture Notes in Comput. Sci.*, 2001, **2178**, 405–420. Springer-Verlag, Berlin.

Rothfuß, R. and Zeitz, M. A toolbox for symbolic nonlinear feedback design. In *Proceedings of the 13th World Congress, International Federation of Automatic Control. Vol. F. Nonlinear Systems II* (Gertler, J. J., Cruz, J. B., Jr., Peshkin, M., Bitmead, R., and Isidori, A., eds). Pergamon, Oxford, 1997, 283–288.

Mittelineaarsete juhtimissüsteemide ülesannete lahendamine webMathematica abil

Maris Tõnso, Heli Rennik ja Ülle Kotta

On tutvustatud TTÜ Küberneetika Instituudis programmeeritud Mathematica-paketi NLControl funktsioone, mis on webMathematica programmeerimisvahendite abil veebis avalikustatud.

NLControl-pakett on mõeldud mittelineaarsete juhtimissüsteemide modelleerimis-, analüüsi- ja sünteesiülesannete lahendamiseks. WebMathematica võimaldab programmeeritud funktsioonid teha veebis kättesaadavaks ilma originaalset programmikoodi avalikustamata. Sellise veebisaidi eesmärgiks on programmide kättesaadavaks tegemine teadlastele üle maailma, aga ka nende kasutamine kraadiõppekursustel. WebMathematica veebilehtede programmeerimiseks on kasutatud HTML-i, Javat ja JavaScripti. Artiklis on kirjeldatud 13 funktsiooni, mis on mõeldud kasutamiseks ajas diskreetsete süsteemide jaoks. Süsteem võib olla kirjeldatud kas olekuvõrrandite või sisend-väljundvõrranditega. Veebilehele on valitud valdavalt funktsioonid, mis baseeruvad diferentsiaalsete 1-vormide alamruumide jadal $\mathscr{H}_k$. Samuti on lühidalt kirjeldatud NLControl-paketi ja veebilehe ülesehitust.

Veebileht on leitav aadressilt http://webmathematica.cc.ioc.ee/webmathematica/NLControl/