Proceedings of the Estonian Academy of Sciences,
2017, **66**, 1, 89–107
https://doi.org/10.3176/proc.2017.1.06
Available online at www.eap.ee/proceedings

CONTROL THEORY

Functions' algebra in nonlinear control: computational aspects and software

Juri Belikov[a], Arvo Kaldmäe[b], Vadim Kaparin[b], Ülle Kotta[b], Alexey Ye. Shumsky[c], Maris Tõnso[b]*, and Alexey Zhirabok[c]

[a] Faculty of Mechanical Engineering, Technion Institute of Technology, Haifa 3200003, Israel; juri.belikov@ttu.ee
[b] Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, 12618 Tallinn, Estonia; vkaparin, arvo, kotta}@cc.ioc.ee
[c] Far Eastern Federal University, Sukhanov Street 8, 690990 Vladivostok, Russia; shumsky@mail.primorye.ru, zhirabok@mail.ru

Received 29 June 2016, revised 10 November 2016, accepted 11 November 2016, available online 2 March 2017

Abstract. The paper describes the *Mathematica*-based software for studying nonlinear control systems. The software relies on an algebraic method, called functions' algebra. The advantage of this approach, over well-known linear algebraic and differential geometric methods is that it is applicable to certain non-smooth systems. The drawback is that the computations are more complicated since the approach manipulates directly with the functions related to the system equations and not with the differential one-forms/vector fields that simplify (linearize) the computations. We have implemented the basic operations of functions' algebra, i.e., partial order, equivalence, summation, and multiplication, but also finding the simplest representative of an equivalence class. The next group of functions is related to the control system and involves binary relation, operators **m**, **M**, and computation of certain sequences of invariant vector functions on the basis of system equations. Finally, we have developed *Mathematica* functions, allowing us to solve the following control problems in case of discrete-time systems: checking accessibility, static state feedback linearization, and disturbance decoupling.

Key words: nonlinear control systems, discrete-time systems, functions' algebra, symbolic computation.

1. INTRODUCTION

The algebraic approach, called historically functions' algebra [13] and based actually on algebraic lattice structure, is developed by analogy with the algebra of partitions [4]. The advantage of this method over difference/linear algebraic and differential geometric methods in nonlinear control is that it allows, in theory, handling also certain types of non-smooth functions. We have specified the class of piecewise smooth functions, which includes, for instance, saturation, friction, and dead zone but not hysteresis and backlash. Although the methodology can, in principle, handle more general functions, computations for such cases become even more complicated without much practical value, since in most 'real-life' examples the functions are piecewise smooth. In particular, the absolute value and signum functions are the most typical non-smooth functions encountered in system descriptions.

* Corresponding author, maris@cc.ioc.ee

However, the possibility of handling non-smooth functions is difficult to realize, since the constructions of the theory are complicated and mainly unknown to control community. Therefore, to support further research within the new framework and explore/evaluate its advantages/disadvantages, the respective symbolic software would be of great help. This paper describes such software, developed within the *Mathematica* environment and incorporated into the package NLControl, which has been developed in the Institute of Cybernetics at Tallinn University of Technology. In certain steps the new *Mathematica* functions that handle computations related to functions' algebra, use the existing functions in NLControl. The first steps towards developing such software were reported in [8] and [10]. Compared to these conference papers, here numerous new algorithms are presented, including (1) computation of the summation operation $\oplus$, (2) computation of the operator $\mathbf{M}$ in the general case, and (3) finding the simplest representative of the equivalence class. Moreover, the solution of the disturbance decoupling problem was not discussed in the conference papers. Note that in this paper we focus on the discrete-time case, though the analogous constructions with some modifications exist for continuous-time systems, too. We comment on the continuous-time case only briefly at the end of the paper.

Unlike the differential geometric [5,12] and difference/linear algebraic methods [1,3], this approach is not based on the globally linearized system description (i.e., on differential one-forms), or on vector fields, defined in the tangent space of directional derivatives, but manipulates directly with the functions, including the state transition map. Therefore, in finding the solutions of different modelling, analysis, and synthesis problems, there is no need to solve specific partial differential equations (PDEs) or integrate one-forms, in principle, though in case of smooth or analytic functions the computations may be handled with the help of distributions and vector spaces over the field of meromorphic functions (codistributions), respectively. On the other hand, since computations are not linearized by applying the differential operators, they are much more complicated. One may hardly expect to find the solutions manually, except for very simple cases. Therefore, for the algebra of functions to be truly applicable, one has to formalize the computations with the key elements of this approach and implement the corresponding algorithms.

The paper is organized as follows. In Section 2 the basic notions of functions' algebra are recalled, as well as the solution of a few nonlinear control problems, based on functions' algebra. In Section 3 the main computation algorithms are given and the details of the *Mathematica* implementation are described. Section 4 is devoted to finding the simplest representative of the equivalence class and in Section 5 the case of non-smooth functions is discussed. In Section 6 two examples are given, demonstrating the application of the developed software to nonlinear control problems. Finally, Section 7 enlightens the possible future studies.

2. MATHEMATICAL SETTING

2.1. Control system and differential one-forms

Consider the discrete-time nonlinear systems, described by the state equations

$$x(t+1) = f(x(t), u(t)), \tag{1}$$

where the state $x(t) \in X \subseteq \mathbb{R}^n$ and the control input $u(t) \in U \subseteq \mathbb{R}^m$.

Hereinafter we use for a variable $\xi(t)$ the notation ξ; the forward shift $\xi(t+1)$ is denoted by ξ^+ and the backward shift $\xi(t-1)$ by ξ^-.

For smooth functions the following approach of differential one-forms proves to be useful in several cases. For smooth $\alpha = [\alpha_1, \dots, \alpha_k]^{\mathrm{T}}$, $\beta = [\beta_1, \dots, \beta_\ell]^{\mathrm{T}}$, depending on variable x, one can define the codistributions Ω_α and Ω_β, respectively, as

$$\Omega_\alpha := \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\alpha_i,\ i = 1, \dots, k\},\ \ \Omega_\beta := \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\beta_\iota,\ \iota = 1, \dots, \ell\}.$$

The symbols α_i, β_ι denote the components of α and β, respectively. The notation $\mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\alpha_i\}$ means that Ω_α contains all linear combinations of one-forms $\mathrm{d}\alpha_i$, computed over a certain field $\mathscr{K}$, described below. Hereinafter we sometimes omit the indices and write $\Omega_\alpha = \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\alpha\}$, $\Omega_\beta = \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\beta\}$. By introducing the column vector $\mathrm{d}x := [\mathrm{d}x_1, \ldots, \mathrm{d}x_n]^{\mathrm{T}}$ and the matrices

$$A := [\alpha_{ij}] := \left[\frac{\partial \alpha_i}{\partial x_j}\right],\ B := [\beta_{\iota j}] := \left[\frac{\partial \beta_\iota}{\partial x_j}\right], \tag{2}$$

where $i = 1, \ldots, k$, $j = 1, \ldots, n$, $\iota = 1, \ldots, \ell$, we write

$$\mathrm{d}\alpha = A\mathrm{d}x,\ \mathrm{d}\beta = B\mathrm{d}x.$$

The field $\mathscr{K}$ is associated with the control system (1), its construction is borrowed from [1]. By $\mathscr{K}$ we denote the field of meromorphic functions in a finite number of variables from the infinite set $\{x,\ u_j(t),\ z_j(-\ell),\ j = 1, \ldots, m,\ t \geqslant 0, \ell > 0\}$. The variables $z_1, \ldots, z_m$ are chosen from the set $\{x, u_j\}$ such that one could locally express the variables x, u_j in terms of x^+ and z_j from (1), i.e. to find a function $(x,u) = \psi(x^+, z)$. This is possible under the assumption that $\mathrm{rank}(\partial f/\partial(x,u)) = n$. The choice of z is not unique, however, all choices lead to the equivalent field. The forward-shift operator $\sigma : \mathscr{K} \to \mathscr{K}$ is defined by $\sigma\varphi(x,u) = \varphi(f(x,u), u^+)$, where f is determined by system (1). The inverse operator of σ is called backward shift and denoted as $\sigma^{-1} : \mathscr{K} \to \mathscr{K}$. Sometimes the abridged notations $\varphi^+(\cdot) = \sigma\varphi(\cdot)$ and $\varphi^-(\cdot) = \sigma^{-1}\varphi(\cdot)$ for $\varphi \in \mathscr{K}$ are used.

2.2. Basics of functions' algebra

In functions' algebra we work with the vectors (of any finite dimension), whose elements are functions defined either on the domain $X \times U$ or on X. Denote the corresponding sets of vectors by $S_{X\times U}$ and S_X, respectively. The elements of $S_{X\times U}$ or S_X are called vector functions.

Below we define two preorders $\leqslant$ and $\leqslant_s$, where $\leqslant_s$ is a special case[1] of $\leqslant$. Note that a preorder is a binary relation that is reflexive and transitive. If, in addition, the relation is also antisymmetric, it is called the relation of partial order. For simplicity, these preorders are defined on S_X. For $S_{X\times U}$, they are defined in a similar manner. Note that the phrase 'for every $x \in X$' throughout this paper should be understood as 'for every $x \in X$ from the intersection of the domains of α and β'.

Definition 1. *Given $\alpha, \beta \in S_X$, one says that*
(i) $\alpha \leqslant \beta$, if for every $x \in X$ there exists a function γ, such that $\beta = \gamma(\alpha)$;
(ii) if the function γ in (i) is defined uniquely for all $x \in X$, we say that $\alpha \leqslant_s \beta$.

Example 1. This example illustrates the difference between preorders $\leqslant$ and $\leqslant_s$. Let $\alpha = x$ and $\beta = x^2$. Then, since $\beta = \alpha^2$ for every $x \in \mathbb{R}$, it is clear that $\alpha \leqslant_s \beta$. But the opposite is not true, because there does not exist a unique function γ such that $\alpha = \gamma(\beta)$ for all $x \in \mathbb{R}$. In fact, if $x < 0$, then $\alpha = -\sqrt{\beta}$ and if $x \geqslant 0$, then $\alpha = \sqrt{\beta}$. However, by definition, $\beta \leqslant \alpha$, since for every $x \in \mathbb{R}$ one can find a function γ such that $\alpha = \gamma(\beta)$.

Based on the preorders $\leqslant$ and $\leqslant_s$, one can define the equivalence relations[2] $\cong$ and $\cong_s$, respectively. In the following we continue only with preorder $\leqslant$, since we use mostly $\leqslant$ when studying the systems of the form (1). The case when $\leqslant_s$ is considered is defined similarly.

[1] Here s stands for 'strong'.
[2] An equivalence relation is a preorder which is also symmetric.

Definition 2. *Vector functions $\alpha, \beta \in S_X$ satisfy the relation $\cong$, if $\alpha \leqslant \beta$ and $\beta \leqslant \alpha$.*

The relation $\cong$ is obviously an equivalence relation.

Example 2. (i) $x \cong x^2, x \not\cong_s x^2$; (ii) $x \cong |x|, x \not\cong_s |x|$; (iii) $|x| \cong x^2$, $|x| \cong_s x^2$; (iv) $[x_1, x_2]^{\mathrm{T}} \cong [x_1, \frac{x_1}{x_2}]^{\mathrm{T}}$, $[x_1, x_2]^{\mathrm{T}} \cong_s [x_1, \frac{x_1}{x_2}]^{\mathrm{T}}$.

If $\alpha \not\leqslant \beta$ and $\beta \not\leqslant \alpha$, then α and β are said to be incomparable.

Example 3. Let $\alpha = [x_1 + x_2, x_3]^{\mathrm{T}}$ and $\beta = [x_2 + x_3]$. Clearly, α and β are incomparable.

The relation $\cong$ is an equivalence relation and thus divides the elements of S_X into the equivalence classes. Let $S_X \backslash \cong$ be the set of the equivalence classes. The relation $\leqslant$ was defined on the set S_X, but can be viewed[3] also as a relation on $S_X \backslash \cong$, where it becomes a partial order. Then the pair $(S_X \backslash \cong, \leqslant)$ becomes a lattice, since $\mathbf{0} := [x_1, \ldots, x_n] \leqslant \alpha \leqslant \mathbf{1}$ for all $\alpha \in S_X \backslash \cong$, where $\mathbf{1}$ is the equivalence class containing constant vector functions.

Remark 1. In control theory it is customary to operate with representatives of some equivalent objects. In the rest of this paper, if not stated otherwise, the equivalence classes are defined by their representatives, which are vector functions from S_X or $S_{X \times U}$ and thus, a vector function should always be understood as an equivalence class. The operations $\times$, $\oplus$, $\mathbf{m}$, $\mathbf{M}$ and the relation Δ are defined on $S_X \backslash \cong$, although in terms of the representatives of the equivalence classes. Also, since we work with equivalence classes, the sign '$=$' should be understood as '$\cong$'.

As $(S_X \backslash \cong, \leqslant)$ is a lattice, we can define the binary operations $\times$ and $\oplus$ as

$$\alpha \times \beta = \inf(\alpha, \beta), \quad \alpha \oplus \beta = \sup(\alpha, \beta) \tag{3}$$

for all $\alpha, \beta \in S_X \backslash \cong$. The infimum and supremum in (3) are considered with respect to the partial order relation $\leqslant$. In the same way, the notion maximal/minimal vector function means maximality/minimality with respect to $\leqslant$.

Example 4. Consider the vector functions $\alpha = [x_1 + x_2, x_3]^{\mathrm{T}}$, $\beta = [x_1 x_3, x_2 x_3]^{\mathrm{T}}$ and let $n = 3$. By definition, $\alpha \times \beta$ is the maximal vector function (containing the minimal number of functionally independent elements) γ that satisfies $\gamma \leqslant \alpha$ and $\gamma \leqslant \beta$. This example contains only one vector function, $\gamma = [x_1, x_2, x_3]^{\mathrm{T}}$, which satisfies $\gamma \leqslant \alpha$ and $\gamma \leqslant \beta$ and thus, $\alpha \times \beta = [x_1, x_2, x_3]^{\mathrm{T}}$.

The vector function $\alpha \oplus \beta$ is defined as the minimal vector function γ satisfying $\alpha \leqslant \gamma$ and $\beta \leqslant \gamma$. In general, there are many vector functions satisfying $\alpha \leqslant \gamma$ and $\beta \leqslant \gamma$. In this example there are two such functions: $\mathbf{1}$ and $(x_1 + x_2)x_3$. Since $(x_1 + x_2)x_3 \leqslant \mathbf{1}$, $\alpha \oplus \beta = (x_1 + x_2)x_3$.

The previously defined lattice will be connected to the system dynamics (1) through the binary relation Δ. Note that Δ is defined only on $S_X \backslash \cong$.

Definition 3. *Given $\alpha, \beta \in S_X \backslash \cong$, one says that (α, β) satisfy binary relation Δ, denoted as $\alpha \Delta \beta$, if for all $x \in X$ and $u \in U$ there exists a function f_* such that*

$$\beta(f(x, u)) = f_*(\alpha(x), u). \tag{4}$$

The binary relation Δ is mostly used for the definition of the operators $\mathbf{m}$ and $\mathbf{M}$.

[3] When an equivalence class is represented by an element (a vector function) of this equivalence class.

Definition 4. *(i)* $\mathbf{m}(\alpha)$ *is a minimal vector function* $\beta \in S_X \backslash \cong$ *that satisfies* $\alpha\Delta\beta$*;*
(ii) $\mathbf{M}(\beta)$ *is a maximal vector function* $\alpha \in S_X \backslash \cong$ *that satisfies* $\alpha\Delta\beta$*.*

Example 5. Consider the system

$$x_1^+ = x_2 u_1, x_2^+ = x_1 + x_3, x_3^+ = x_3 + u_2$$

and the vector function $\alpha = [x_1, x_2]^{\mathrm{T}}$. First, note that $\alpha\Delta x_1$, because x_1^+ can be written in terms of α and u:

$$x_1(f(x,u)) = x_1^+ = \alpha_2 u_1. \tag{5}$$

Also $\alpha\Delta[x_1, x_2 - x_3]^{\mathrm{T}}$, because of (5) and $(x_2 - x_3)^+ = \alpha_1 - u_2$. Note that $[x_1, x_2 - x_3]^{\mathrm{T}} \leqslant x_1$; in fact $[x_1, x_2 - x_3]^{\mathrm{T}}$ is a minimal vector function β satisfying $\alpha\Delta\beta$ because there exist no other functions whose forward shifts do not depend on x_3, and thus

$$\mathbf{m}(\alpha) = [x_1, x_2 - x_3]^{\mathrm{T}}.$$

Next, observe that one can always write $\alpha(f(x,u))$ in terms of x and u, which means by Definition 3 that $x\Delta\alpha$. Therefore, $x \leqslant \mathbf{M}(\alpha)$ for every α. In this example, the vector function $\gamma := [x_2, x_1 + x_3]^{\mathrm{T}}$ also satisfies $\gamma\Delta\alpha$ because

$$\alpha_1^+ = \gamma_1 u_1,\ \alpha_2^+ = \gamma_2.$$

In fact, γ is a maximal vector function that satisfies $\gamma\Delta\alpha$ and thus, by Definition 4,

$$\mathbf{M}(\alpha) = \gamma = [x_2, x_1 + x_3]^{\mathrm{T}}.$$

2.3. Applications of functions' algebra

In this section, the solutions of three nonlinear control problems (accessibility, static state feedback linearization, and disturbance decoupling problem) are formulated in terms of functions' algebra. For that purpose, we first define the notion of f-invariant vector function.

Definition 5. *The vector function* δ *is said to be invariant with respect to the system dynamics (1), or said alternatively,* f*-invariant, if* $\delta\Delta\delta$*.*

Next, one has to find a minimal (containing the maximal number of functionally independent components) vector function $\delta^1(x)$ such that its forward shift $\delta^1(x^+) = \delta^1(f(x,u))$ does not depend on the control variable u. If f is smooth, $\delta^1(x)$ satisfies the condition $\frac{\partial}{\partial u}\delta^1(f(x,u)) \equiv 0$. The vector function $\delta^1(x)$ initializes Algorithm 1 below. In general, the components of $\delta^1(x)$ are scalar functions with relative degree two or more [5].

Algorithm 1 finds the minimal f-invariant vector function δ, satisfying the condition $\delta^1 \leqslant \delta$.

Algorithm 1 ([11])**.** Given δ^1, compute recursively for $i \geqslant 1$, using the formula

$$\delta^{i+1} = \delta^i \oplus \mathbf{m}(\delta^i),$$

the sequence of non-decreasing vector functions $\delta^1 \leqslant \delta^2 \leqslant \cdots \leqslant \delta^i \leqslant \cdots$. Then there exists a finite j such that $\delta^j \ncong \delta^{j-1}$ but $\delta^{j+l} \cong \delta^j$, for all $l \geqslant 1$. Define $\delta := \delta^j$.

A non-constant vector function $\alpha \in S_X$ is said to be an *autonomous variable* of system (1) if there exist a non-constant function F and an integer $\mu \geqslant 1$ such that $F(\alpha, \sigma(\alpha), \ldots, \sigma^\mu(\alpha)) = 0$. System (1) is said to be *accessible* if it has no autonomous variable.

Theorem 6 ([11]). *System (1) is accessible iff for some k, $\delta^{k-1} \cong \delta^k = \mathbf{1}$, i.e., the minimal f-invariant vector function, satisfying the condition $\delta^1 \leqslant \delta$, is constant.*

System (1) is said to be *static state feedback linearizable* if, generically, there exist (i) a state coordinate transformation $\varphi : X \to X$ and (ii) a regular static state feedback of the form $u = \vartheta(x,v)$ such that, in the new coordinates $z = \varphi(x)$, the compensated system reads $z_i^+ = z_{i+1}$, for $i = 1,\dots,n-1$ and $z_n^+ = v$.

Theorem 7 ([11]). *The single-input system (1) is static state feedback linearizable iff $\delta^i \neq \mathbf{1}$, for $i = 1,\dots,n-1$, but $\delta^n = \mathbf{1}$. The state transformation is given by $z_1 = \delta^{n-1}$, $z_{i+1} = \mathbf{M}^{i-1}(\delta^{n-1})$ for $i = 2,\dots,n$.*

The third problem under study is the disturbance decoupling problem under a dynamic measurement feedback (DDDPM). Consider a discrete-time nonlinear control system

$$x(t+1) = f(x(t),u(t),w(t)), \;\; y(t) = h(x(t)), \;\; y_*(t) = h_*(x(t)), \tag{6}$$

where $x(t) \in X \subseteq \mathbb{R}^n$ is the state, $u(t) \in U \subseteq \mathbb{R}^m$ is the control, $w(t) \in W \subseteq \mathbb{R}^p$ is the unmeasurable disturbance, $y(t) \in Y \subseteq \mathbb{R}^l$ is the measured output, and $y_*(t) \in Y_* \subseteq \mathbb{R}^L$ is the output-to-be-controlled. The DDDPM can be stated as follows: find a dynamic measurement feedback of the form

$$z(t+1) = F(z(t),y(t),v(t)), \;\; u(t) = G(z(t),y(t),v(t)), \tag{7}$$

where $v(t) \in V \subseteq \mathbb{R}^m$ and

$$\operatorname{rank}[\partial G/\partial v] = m, \tag{8}$$

such that the values of the outputs-to-be-controlled $y_*(t)$, for $t \geqslant 0$, of the closed-loop system are independent of the disturbances w. Note that we call the compensator, described by (7), regular if it generically defines the (y,z)-dependent one-to-one correspondence between the variables v and u. One says that the disturbance decoupling problem is solvable via static output feedback if $u = G(y,v)$.

Find first a minimal vector function $\alpha_0(x)$ such that its forward shift $\alpha_0(f(x,u,w))$ does not depend on the unmeasurable disturbance w.

Definition 8. *The vector function α is said to be(h,f)-invariant if $(\alpha \times h)\Delta\alpha$.*

Definition 9. *The vector function α is said to be a controlled invariant if there exists a regular static state feedback $u = G(x,v)$ such that the vector function α is f-invariant for the closed-loop system.*

Theorem 10 ([6]). *System (6) can be disturbance decoupled by feedback (7), where $z = \alpha(x)$ iff there exists an (h,f)-invariant vector function ϕ and a controlled invariant vector function ξ such that $\alpha_0 \leqslant \phi \leqslant \xi \leqslant h_*$.*

3. *MATHEMATICA* IMPLEMENTATION

The programs, described in this section, are implemented as a part of the *Mathematica*-based package NLControl. By employing NLControl, various problems, related to analysis, synthesis, and modelling of nonlinear control systems can be solved. The most important functions of the package are made available on the package website [2] using *webMathematica* technology, so that anyone can use them via a internet browser without installing special software.

Here the methods are described to compute the operations and operators, defined in the previous section. To simplify the presentation, assume first that the vector functions α and β are smooth. The non-smooth case is discussed later. This paper focuses only on the case of the partial order relation $\alpha \leqslant \beta$.

For the strong relation $\alpha \leqslant_s \beta$, in Definition 1 the existence of unique γ is required for all $x \in X$. As demonstrated in Example 1, uniqueness essentially depends on the chosen region. Thus, the study of strong relation basically requires determining whether a certain equation (or system of equations) has a unique solution in a certain region. The other possible task is to determine a ‘good’ region where unique γ exists.

3.1. Relation of partial order

A condition to check whether $\alpha \leqslant \beta$ is formulated as follows.

Proposition 11. *For smooth vector functions $\alpha, \beta \in S_X \backslash \cong$, $\alpha \leqslant \beta$ if and only if*

$$\operatorname{rank}_{\mathscr{K}}\left[\frac{\partial \alpha}{\partial x}\right] = \operatorname{rank}_{\mathscr{K}}\left[\left(\frac{\partial \alpha}{\partial x}\right)^{\mathrm{T}}, \left(\frac{\partial \beta}{\partial x}\right)^{\mathrm{T}}\right]^{\mathrm{T}} \tag{9}$$

for all $x(t) \in X$.

Proof.

$$\beta(x) = \gamma(\alpha(x)) \quad \Leftrightarrow \quad \frac{\partial \beta}{\partial x} = \frac{\partial \gamma}{\partial \alpha}\frac{\partial \alpha}{\partial x} \quad \Leftrightarrow \quad (9).$$

□

Example 6. Consider the vector functions $\alpha = [x_1x_2, x_3]^{\mathrm{T}}$ and $\beta = [x_1 + x_3, x_2]^{\mathrm{T}}$ from S_X. The package can be loaded by the command[4]

```
In[1]:= <<NLControl`Master`
```

In this implementation the column vectors α and β are represented in the single pair of brackets[5] for the sake of simplicity:

```
In[2]:= α = {x1 x2, x3}; β = {x1 + x3, x2}; xi = {x1, x2, x3};
```

It is also necessary to specify the list of the variables x_i, denoted by **xi**. The preorder of α and β can now be checked by

```
In[3]:= PartialPreOrder[α, β, xi]

Out[3]= False
```

which means that $\alpha \nleqslant \beta$. The function **PartialPreOrder** is based on the rank condition (9); *Mathematica* function **MatrixRank** is used to compute the ranks. The function **Incomparability** checks if α and β are incomparable:

```
In[4]:= Incomparability[α, β, xi]

Out[4]= True
```

If we look at the vector functions α and β as representatives of the equivalence classes, the same function can be used to check whether they satisfy the partial order $\leqslant$.

3.2. Equivalence

The equivalence of two vector functions α and β can be checked easily, by just examining whether $\alpha \leqslant \beta$ and $\beta \leqslant \alpha$. A function **LatticeEquivalent[α, β, xi]** allows one to check if α and β belong to the same equivalence class.

[4] For the sake of compactness, the *Mathematica* lines are printed in roman font rather than in typewriter font, appearing in the original *Mathematica* notebooks.

[5] In *Mathematica* the single brackets typically indicate the row vector.

3.3. Computation of $\boldsymbol{\alpha \times \beta}$

This operation can be computed by a simple formula

$$\alpha \times \beta = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

Note that typically a simpler representative of the equivalence class exists, since there may be dependent elements in $[\alpha^{\mathrm{T}}, \beta^{\mathrm{T}}]^{\mathrm{T}}$ (see Section 4 for details):

```
In[5]:= α = {x1 + x2, x3}; β = {x1 x3, x2 x3}; xi = {x1, x2, x3};
        LatticeTimes[α, β, xi]

Out[5]= {x1, x2, x3}
```

3.4. Computation of $\boldsymbol{\alpha \oplus \beta}$

To find $\alpha \oplus \beta$, the following algorithm can be used:

Algorithm 2.

1. Compute $\Omega = \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\alpha\} \cap \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\beta\}$.
2. Find the largest integrable subspace $\hat{\Omega}$ of Ω.
3. Integration of $\hat{\Omega}$ gives the components of the vector function $\alpha \oplus \beta$.

Intersection $\mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\alpha\} \cap \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}\beta\}$ can be computed by the following straightforward algorithm:

Algorithm 3.

1. Compute
$$K := \ker \begin{bmatrix} A \\ B \end{bmatrix}^{\mathrm{T}},$$
where matrices A and B are defined by (2) and ker denotes the null space of the matrix.
2. Take the first k columns of K and denote the obtained matrix by $\bar{K}$.
3. Intersection $\mathrm{sp}\{\mathrm{d}\alpha\} \cap \mathrm{sp}\{\mathrm{d}\beta\} = \mathrm{sp}\{\bar{K}A\mathrm{d}x\}$, where $\bar{K}A$ is the product of matrices.

The largest integrable subspace of $\Omega = \mathrm{sp}_{\mathscr{K}}\{\omega_1, \ldots, \omega_k\}$ can be computed as the limit of the following sequence of subspaces, sometimes known as the derived flag of the Pfaffian system (see, for instance, [1]).

Algorithm 4.

$$\begin{aligned} I_0 &= \mathrm{sp}_{\mathscr{K}}\{\omega_1, \ldots, \omega_k\}, \\ I_j &= \mathrm{sp}_{\mathscr{K}}\{\omega \in I_{j-1} | \mathrm{d}\omega = 0 \mod I_{j-1}\}. \end{aligned} \tag{10}$$

Note that $\mathrm{d}\omega = 0 \bmod \mathrm{span}_{\mathscr{K}}\{\omega_1, \ldots, \omega_p\}$ means $\mathrm{d}\omega \wedge \omega_1 \wedge \cdots \wedge \omega_p = 0$.

The function **LatticePlus[α, β, xi]** finds $\alpha \oplus \beta$, where **xi** is a list of variables.

3.5. Binary relation Δ

The equality (4) may be alternatively expressed as follows:

$$\alpha(x) \times u \leqslant \beta(f(x,u)). \tag{11}$$

Indeed, by applying the definition of partial order to (11) and replacing γ by f_* we get the equality (4). The condition (11) is more suitable for computer implementation than (4) since it does not depend on the unknown vector function f_* and, moreover, it allows one to take advantage of the already existing function `PartialPreOrder`. The *Mathematica* function allowing one to check whether α and β satisfy the binary relation Δ is called `BinaryRelationΔ`.

Example 7. Consider the discrete-time system

$$x_1^+ = x_2x_3 + x_1, x_2^+ = x_3 + u_2, x_3^+ = u_2, x_4^+ = x_3x_4 + x_1u_1. \tag{12}$$

The state equations may be entered in the form[6]:

```
In[6]:= f = {x2 x3 + x1, x3 + u2, u2, x3 x4 + x1 u1};
        Xt = {x1, x2, x3, x4}; Ut = {u1, u2};
        steq = StateSpace[f, Xt, Ut, t, Shift];
```

Suppose that $\alpha = [x_1, x_2, x_3]^{\mathrm{T}}$ and $\beta = [x_1 + x_2]$:

```
In[7]:= α = {x1, x2, x3}; β = {x1 + x2};
```

Clearly, $\beta(f(x,u)) = [x_1^+ + x_2^+] = [x_2x_3 + x_1 + x_3 + u_2]$ is computable from $\alpha = [\alpha_1, \alpha_2, \alpha_3]^{\mathrm{T}}$ and u as $[\alpha_2\alpha_3 + \alpha_1 + \alpha_3 + u_2]$. Therefore, $\alpha\Delta\beta$, i.e.,

```
In[8]:= BinaryRelationΔ[α, β]

Out[8]= True
```

Assume now that $\tilde{\alpha} = [x_2x_3 + x_1, x_3]^{\mathrm{T}}$. Clearly, $\beta(f(x,u)) = [x_2x_3 + x_1 + x_3 + u_2] = [\tilde{\alpha}_1 + \tilde{\alpha}_2 + u_2]$ is computable from the knowledge of $\tilde{\alpha}$. Therefore,

```
In[9]:= α̃ = {x2 x3 + x1, x3}; BinaryRelationΔ[α̃, β]

Out[9]= True
```

Finally, if $\bar{\alpha} = [x_2x_3 + x_1 + x_3]$, then $\beta(f(x,u)) = [\bar{\alpha}_1 + u_2]$. Therefore $\alpha_2\Delta\beta$:

```
In[10]:= ᾱ = {x2 x3 + x1 + x3}; BinaryRelationΔ[ᾱ, β]

Out[10]= True
```

3.6. Operator m

By Definition 3 and (11), the condition $\mathbf{m}(\alpha)(f) \geqslant \alpha \times u$ must be satisfied for vector function α. Moreover, since $[\zeta(x)]^+ = \zeta(f)$ for any function ζ, obviously also $\mathbf{m}(\alpha)(f) \geqslant f$. From the above and the definition of the operator $\oplus$,

$$\mathbf{m}(\alpha)(f) = (\alpha \times u) \oplus f. \tag{13}$$

Finally, observe that the left-hand side of (13) is the forward shift of $\mathbf{m}(\alpha)$ and thus,

$$\mathbf{m}(\alpha) = [(\alpha \times u) \oplus f]^-. \tag{14}$$

The program to compute the operator **m** is based on the formula (14).

Example 8. Consider again system (12). Let $\alpha(x) = [x_1, (x_2 + x_4)x_3]^{\mathrm{T}}$. To compute $\mathbf{m}(\alpha)$, the vector function $(\alpha \times u) \oplus f$ has to be found:

```
In[11]:= α ={x1, (x2 + x4) x3}; vars = {x1, x2, x3, x4, u1, u2};
         mαPlus = LatticePlus[LatticeTimes[α, Ut, vars], f, vars]

Out[11]= {u2, (1 + u1) x1 + x3 (x2 + x4)}
```

To get $\mathbf{m}(\alpha)$, we have to find the backward shift of the previous result. In general, backward shifts of the variables $x_i^- := x_i(t-1)$, $i = 1, \ldots, n$ and $u_j^- := u_j(t-1)$, $j = 1, \ldots, m$ are defined by system equations (12) and for $\alpha(x,u) \in S_X$ the backward shift $\alpha(x,u)^- := \alpha(x^-, u^-)$. Note that not all variables x_i^-, $i = 1, \ldots, n$ and u_j^-, $j = 1, \ldots, m$ are independent. Alternative choices are possible for selecting m independent variables.

[6] For the sake of compactness we have omitted the obligatory argument t. In the original *Mathematica* files all variables $x_1, x_2, \ldots, u_1, u_2, \ldots, y_1, y_2, \ldots$, should be written as $x_1[t], x_2[t], \ldots, u_1[t], u_2[t], \ldots, y_1[t], y_2[t], \ldots$, as long as they are related to the state equations.

Choose, for instance, x_1^- and u_1^- as independent variables[7]. Then the remaining variables may be obtained by solving (12) for x_2, x_3, x_4, u_2 and applying to them the backward shift operator

$$x_2^- = \frac{x_1 - x_1^-}{x_2 - x_3},\ x_3^- = x_2 - x_3,\ x_4^- = \frac{x_4 - u_1^- x_1^-}{x_2 - x_3},\ u_2^- = x_3.$$

Then $((\alpha \times u) \oplus f)^- \cong [u_2^-, (x_2^- + x_4^-)x_3^- + x_1^- + x_1^- u_1^-]^{\mathrm{T}} = [x_3, x_1 + x_4]^{\mathrm{T}}$. The NLControl function **BackwardShift** has been developed earlier:

```
In[12]:= BackwardShift[mαPlus, steq]
Out[12]= {x3, x1 + x4}
```

The function **OperatorSmallM** finds $\mathbf{m}(\alpha)$ at once:

```
In[13]:= OperatorSmallM[α, steq]
Out[13]= {x3, x1 + x4}
```

However, if $\alpha(x) = [x_3]$, we get

```
In[14]:= OperatorSmallM[{x3}, steq]
Out[14]= {x2, x3}
```

3.7. Operator M

Unlike for $\mathbf{m}(\alpha)$, there is no general formula to compute $\mathbf{M}(\alpha)$ in terms of $\leqslant$, $\times$, and $\oplus$. In some special cases one can give a formula for $\mathbf{M}(\alpha)$. For example, consider the case when the components $\alpha_j(f(x,u))$, $j = 1, \ldots, k$, of $\alpha(f(x,u))$ can be represented in the form

$$\alpha_j(f(x,u)) = \sum_{i=1}^{d_j} a_{ji}(x) b_{ji}(u), \tag{15}$$

where $a_{j1}(x), \ldots, a_{jd_j}(x)$ are arbitrary functions and all the functions $b_{j1}(u), \ldots, b_{jd_j}(u)$ that are non-constant are linearly independent. Then

$$\mathbf{M}(\alpha) := \prod_{j=1}^{k} a_{j1} \times \cdots \times a_{jd_j}. \tag{16}$$

Technically the most complicated step is the decomposition of the expression $\alpha_j(f(x,u))$ in (15) into the sum of products $a_{ji}(x)$ and $b_{ji}(u)$. For that purpose the products and positive integer powers are expanded out: for instance, $(x+u)^2$ is rewritten in the form $x^2 + 2xu + u^2$. If $\alpha(f(x,u))$ contains rational terms, the denominators are factorized: for instance, $\frac{1}{2+2u+x+ux}$ is rewritten as $\frac{1}{x+2} \cdot \frac{1}{u+1}$. Complex roots are ignored. Additionally, the functions with arguments involving the sum of x and u are written in the form of a product: for instance, e^{x+u} is replaced by $e^x \cdot e^u$ and $\sin(x+u)$ by $\cos x \sin u + \cos u \sin x$. Finally, each summand of $\alpha_j(f(x,u))$ is split into two parts: $a_{ji}(x)$ and $b_{ji}(u)$. However, there exist simple expressions, for instance, $1/(x+u)$, which cannot be decomposed. In such cases the following general algorithm has to be applied.

If (15) does not hold, one may use the general Algorithm 5 for computing $\mathbf{M}(\alpha)$, expressed in terms of one-forms.

Algorithm 5. Let $\alpha = [\alpha_1, \ldots, \alpha_k]^{\mathrm{T}}$.
1. Find the one-forms ω_i, $i = 1, \ldots, k$, such that $\mathrm{d}\alpha_i(f) = \omega_i + \bar{\omega}_i$, where $\omega_i \in \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}x\}$ and $\bar{\omega}_i \in \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}u\}$.
2. Compute the minimal integrable subspace Ω that contains $\mathrm{sp}_{\mathscr{K}}\{\omega_1, \ldots, \omega_k\}$.
3. The integration of Ω gives the elements of vector $\mathbf{M}(\alpha)$.

[7] One can consult [1] for finding the alternative choices of independent variables.

To justify Algorithm 5, note that by definition, $\mathbf{M}(\alpha)$ is a maximal vector function ξ that satisfies

$$\alpha(f) = f_*(\xi, u) \tag{17}$$

for some f_*, i.e., $\xi \Delta \alpha$. Algorithm 5 computes the integrable subspace Ω such that $\mathrm{d}\alpha(f) \in \Omega + \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}u\}$[8]. From above and Definition 3, integrating Ω yields vector function μ that satisfies $\mu\Delta\alpha$. To make μ maximal (in the sense of $\leqslant$), one has to integrate minimal Ω.

The following algorithm is used to compute the minimal integrable subspace that contains $\mathrm{span}_{\mathscr{K}}\{\omega_1, \dots, \omega_k\}$.

Algorithm 6. Given $\Omega = \mathrm{sp}_{\mathscr{K}}\{\omega_1, \dots, \omega_k\}$, proceed as follows:

1. Compute the dimension of the minimal integrable subspace that contains Ω. This is equal to minimal γ such that
$$(\mathrm{d}\pi)^\gamma \wedge \pi = 0,$$
where $\pi = \sum_{i=1}^k \alpha_i \omega_i$ and $\alpha_i \in \mathscr{K}$ are viewed as new independent variables.
2. Define for $i = 1, \dots, \gamma - k$ the one-forms
$$\bar{\omega}_i = \sum_{j=1}^n a_{ij} \mathrm{d}x_j,$$
where $a_{ij} \in \mathscr{K}$.
3. The one-forms $\bar{\omega}_i$, $i = 1, \dots, \gamma - k$ have to satisfy the condition
$$\mathrm{d}\omega_l \wedge \omega_1 \wedge \dots \wedge \omega_k \wedge \bar{\omega}_1 \wedge \dots \wedge \bar{\omega}_{\gamma-k} = 0,$$
because the subspace $\mathrm{sp}_{\mathscr{K}}\{\omega_1, \dots, \omega_k, \bar{\omega}_1, \dots, \bar{\omega}_{\gamma-k}\}$ has to be integrable. This gives a system of equations in a_{ij}, which has to be solved.
4. It remains to guarantee that $\mathrm{d}\bar{\omega}_i = 0$ for $i = 1, \dots, \gamma - k$, i.e.,
$$\sum_{j=1}^n \mathrm{d}a_{ij} \wedge \mathrm{d}x_j = 0$$
for $i = 1, \dots, \gamma - k$. This gives a system of differential equations, which has to be solved.

The function **OperatorCapitalM [α, steq]** finds $\mathbf{M}(\alpha)$, if **steq** represents the sate equations, defined as in [7].

4. THE SIMPLEST REPRESENTATIVE OF THE CLASS

In order to simplify the computations, one has to replace the components of vector functions by equivalent but simpler functions. The task seems to be trivial at first glance, however, the formalization of it has turned out to be one of the most complicated steps in the implementation. First, simplicity is something that cannot be formally defined as 'α is simpler than β if certain condition is satisfied'. Simplicity is tightly related to the problem under study. In this section we speak about simplicity regarding its intuitive meaning; for instance, we consider $[x_1]$ to be simpler than $[x_1^2]$ and $[x_1, x_2]^\mathrm{T}$ simpler than $[x_1 + x_2, x_1 x_2]^\mathrm{T}$. In practice the simplicity of the expression is measured by the *Mathematica* function **LeafCount**. We developed two approaches for simplification: the first is based on one-forms and the other on the simplification rules. Unfortunately, both approaches have certain classes of functions that cannot be handled.

[8] This is always possible since in the extreme case one may take $\Omega = \mathrm{sp}_{\mathscr{K}}\{\mathrm{d}x\}$.

4.1. Approach of one-forms

Let $\alpha = [\alpha_1, \dots, \alpha_k]^{\mathrm{T}}$ and suppose A is defined by (2). Transform the matrix A into the row-reduced form $\bar{A}$, using linear transformations over $\mathscr{K}$. This can be done by the *Mathematica* built-in function **RowReduce**. If the fractions appear in the matrix $\bar{A}$, the respective rows are multiplied by the least common multiple of their denominators. This leads usually to simpler matrix entries. The zero rows are removed, so $\bar{A}$ is a $\bar{k} \times n$ matrix.

Integrate the one-forms $\omega_i := \sum_{i=1}^{n} \bar{\alpha}_{ij} \mathrm{d}x_j$, $i = 1, \dots, \bar{k}$. This procedure requires solving the system of PDEs, since *Mathematica* (version 10.4) has a limited ability to solve systems of PDEs; the task is reduced to solving a sequence of single PDEs. This process has been implemented earlier in NLControl as a function **IntegrateOneForms**, described in [9].

Example 9. Let $\alpha = [x_1x_2, x_1 + x_2, x_1^2]^{\mathrm{T}}$. The one-forms $\mathrm{d}\alpha = [x_2\mathrm{d}x_1 + x_1\mathrm{d}x_2, \mathrm{d}x_1 + \mathrm{d}x_2, 2x_1\mathrm{d}x_1]^{\mathrm{T}}$. Transforming the matrix $[\alpha_{ij}]$ into the row-echelon form yields $\omega_1 = \mathrm{d}x_1$, $\omega_2 = \mathrm{d}x_2$, thus $\alpha \cong [x_1, x_2]^{\mathrm{T}}$.

However, this approach is not problem-free. Integration often returns the result which is more complex than the original vector.

Example 10. Let $\alpha = [\frac{1}{x_1+x_2^2x_3}, x_1]^{\mathrm{T}}$. A correct simplified vector is obviously $\tilde{\alpha} = [x_2^2x_3, x_1]^{\mathrm{T}}$. The one-forms are

$$\begin{bmatrix}\omega_1\\ \omega_2\end{bmatrix} = \begin{bmatrix} -\frac{1}{(x_1+x_2^2x_3)^2} & -\frac{2x_2x_3}{(x_1+x_2^2x_3)^2} & -\frac{x_2^2}{(x_1+x_2^2x_3)^2} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix}\mathrm{d}x_1\\ \mathrm{d}x_2\\ \mathrm{d}x_3\end{bmatrix}.$$

Null space of $[\alpha_{ij}]$ is $[0, -\frac{x_2}{2x_3}, 1]$, thus we are seeking for the function $X = X(x_1, x_2, x_3)$, being solution of the PDE

$$-\frac{x_2}{2x_3}\frac{\partial X}{\partial x_2} + \frac{\partial X}{\partial x_3} = 0.$$

The *Mathematica* function **DSolve** yields $X = C(x_1, x_2\sqrt{x_3})$, where C is an arbitrary function. Choosing C as the identity function yields the simplified vector $\tilde{\alpha} = [C_1, C_2] = [x_1, x_2\sqrt{x_3}]^{\mathrm{T}}$, which is not the desired solution, because $x_2\sqrt{x_3}$ is more complex than $x_2^2x_3$ and, additionally, the domain of the function has also changed. There are two ideas that may offer a solution in this situation.

First, observe that choosing $\tilde{\alpha} = [C_1, C_2^2]$ would result in the desired solution. The problem here is that though the solution is easy to see through in the case of simple examples, it is hard to generalize into the universal algorithm.

Second, it turns out that reordering the state coordinates also gives the desired solution. The choice $x = (x_1, x_3, x_2)$ causes the second and the third columns of the matrix $[\alpha_{ij}]$ to be swapped and therefore, the null space of the new matrix will be $[0, -\frac{2x_3}{x_2}, 1]$. Solving the respective PDE

$$-\frac{2x_3}{x_2}\frac{\partial X}{\partial x_3} + \frac{\partial X}{\partial x_2} = 0$$

yields $X = C(x_1, x_2^2x_3)$. Unfortunately, again there is no method to determine the suitable order or the coordinates *a priori*.

Moreover, sometimes *Mathematica* fails to solve the PDEs, even if the solution exists.

Example 11. Let $\alpha = [x_1 + x_2x_3 + x_4, x_1x_2 + x_3, x_4]^{\mathrm{T}}$. This vector can be clearly simplified. The variable x_4 appears independently as an element of the vector, thus

$$\alpha \cong [x_1 + x_2x_3, x_1x_2 + x_3, x_4]^{\mathrm{T}}. \tag{18}$$

However, if we find the matrix

$$A = \begin{bmatrix} 1 & x_3 & x_2 & 1 \\ x_2 & x_1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

compute its null space $\left[\frac{x_3 - x_1 x_2}{x_1 - x_2 x_3}, \frac{x_2^2 - 1}{x_1 - x_2 x_3}, 1, 0\right]$, we have to solve the PDE

$$\left(\frac{x_3 - x_1 x_2}{x_1 - x_2 x_3}\right)\frac{\partial X}{\partial x_1} + \left(\frac{x_2^2 - 1}{x_1 - x_2 x_3}\right)\frac{\partial X}{\partial x_2} + \frac{\partial X}{\partial x_3} = 0.$$

Mathematica is not able to solve this PDE and the simplification program fails. Note that changing the order of the coordinates to $[x_1, x_3, x_2, x_4]$ leads to the PDE

$$\left(\frac{x_1 - x_2 x_3}{x_2^2 - 1}\right)\frac{\partial X}{\partial x_3} + \frac{\partial X}{\partial x_2} + \left(\frac{x_3 - x_1 x_2}{x_2^2 - 1}\right)\frac{\partial X}{\partial x_1} = 0,$$

which *Mathematica* can solve, yielding the vector function (18). However, there is no general method to determine the right order or the coordinates *a priori*.

As a temporary solution we have implemented the program which finds all permutations of coordinates and then tries to solve the respective PDEs in turn until one of them yields the solution. The time limit 0.5 s has been set for each PDE; if the solution is not found within this time, the program turns to the next PDE.

4.2. Replacement rules

This approach uses certain replacement rules, based on theoretical justification. The important advantage of this method is that it is applicable also to non-smooth functions. A few well-known equivalence rules are listed below[9]:

(i) const $\cong \mathbf{1}$,

(ii) $\alpha(x) + \text{const} \cong \alpha(x)$,

(iii) $\alpha^k(x) \cong \alpha(x)$, $k \in \mathbb{Z}$,

(iv) $\sqrt[k]{\alpha(x)} \cong \alpha(x)$, $k \in \mathbb{Z}$.

Generalizing these rules yields the following three simplification methods.

1. Replacing square blocks by identity matrices. Let $x = [x_1, \ldots, x_n]$, $\alpha = [\alpha_1, \ldots, \alpha_n]$. Assume that $\operatorname{rank}[\alpha_{ij}] = n$ in (2) (if non-differentiable functions appear in α, then formal derivation is applied). In such case $\alpha \cong [x_1, \ldots, x_n]$. Note that the square block may appear as a minor of $[\alpha_{ij}]$. The program searches for such minors and replaces them by identity matrices. The minors can be from order 1 up to the order n. For example, in the case of the 2nd-order minor the matrix $[\alpha_{ij}]$ has to be in the form

$$[\alpha_{ij}] = \begin{bmatrix} \cdots & 0 & \cdots & 0 & \cdots \\ 0 & \alpha_{i_1 j_1} & 0 & \alpha_{i_1 j_2} & 0 \\ \cdots & 0 & \cdots & 0 & \cdots \\ 0 & \alpha_{i_2 j_1} & 0 & \alpha_{i_2 j_2} & 0 \\ \cdots & 0 & \cdots & 0 & \cdots \end{bmatrix}. \tag{19}$$

In such a case, if the rank of the minor is 2, we have the equivalence $[\alpha_{i_1}, \alpha_{i_2}]^{\mathrm{T}} \cong [x_{j_1}, x_{j_2}]^{\mathrm{T}}$.

Unfortunately, currently the program does not recognize the minors with zero elements.

2. Reducing operators. The generalization of rules (iii) and (iv) leads to the idea that in principle we may consider $G(g(x)) \cong g(x)$ for any (single-valued) elementary function G. For instance, the functions G,

[9] Rules (iii) and (iv) hold only in case of non-strict equivalence.

involving trigonometric functions, $\log_a g(x)$ and exponent $g^a(x)$, where a is a real number or a function that does not depend on x, may be removed.

3. Reducing sums and products. The third simplification algorithm is based on the observations that

$$\begin{aligned} &[\beta(x)+\gamma(x_1,\dots,x_i),x_1,\dots,x_i]^{\mathrm{T}} \cong [\beta(x),x_1,\dots,x_i]^{\mathrm{T}}, \\ &[\beta(x)\gamma(x_1,\dots,x_i),x_1,\dots,x_i]^{\mathrm{T}} \cong [\beta(x),x_1,\dots,x_i]^{\mathrm{T}}. \end{aligned} \tag{20}$$

If the term $\gamma = \text{const}$, we obtain from (20) the well-known rule (ii). This program is able to efficiently handle Example 11, in which case the integration failed.

This program has much space for improvements. For instance, the vector $[\sin(x_1x_2)x_3, x_1x_2] \cong [x_3, x_1x_2]$. However, currently the program does not recognize the equivalence, since x_1x_2 is a product, not a pure coordinate.

Combining the three methods. The program **LatticeSimplify** applies all three methods in turn repeatedly until the expression no longer changes.

Example 12. (Continuation of Example 10). The application of the 2nd method to $\alpha = [\frac{1}{x_1+x_2^2x_3}, x_1]^{\mathrm{T}}$ yields $\bar{\alpha} = [x_1 + x_2^2x_3, x_1]^{\mathrm{T}}$. The application of the 3rd method to $\bar{\alpha}$ yields the final result $[x_2^2x_3, x_1]^{\mathrm{T}}$.

5. NON-SMOOTH CASE

Regarding the non-smooth functions, the book [13] gives the algorithm to check the relation $\alpha \leqslant \beta$ if β has non-smooth components.

Algorithm 7.

1. Rewrite $\beta(x)$ in the form (this form always exists) $\beta(x) = \bar{\beta}(x, \phi(\zeta(x)))$, where $\bar{\beta}$ and ζ are smooth functions and ϕ is non-smooth.
2. Introduce the additional vector $\bar{x} := \phi(\zeta(x))$.
3. Rewrite β in the 'smooth' form $\beta = \bar{\beta}(x, \bar{x})$.
4. The relation $\alpha \leqslant \beta$ holds if $\alpha \leqslant \zeta$ and $[\alpha^{\mathrm{T}}, \bar{x}^{\mathrm{T}}]^{\mathrm{T}} \leqslant \bar{\beta}$.

The implementation of this algorithm is straightforward.

Unfortunately, there is no theoretical solution for the case when α has non-smooth components; thus, there is no method for checking the equivalence of two vectors. We see two practical approaches, which may offer a solution.

The first approach is based on the notion of equivalence. Regarding the properties of the absolute value and signum function, we can consider the derivatives $\frac{\mathrm{d}}{\mathrm{d}x}|x| \cong \mathrm{sign}x$ and $\frac{\mathrm{d}}{\mathrm{d}x}(\mathrm{sign}x) \cong 0$ if $x \neq 0$. The implementation of this replacement is simple: *Mathematica* denotes, by default, derivatives of $|x|$ and sign$x$ by formal functions **Abs' [x]** and **Sign' [x]**, respectively. All we have to do is to replace these functions by **Sign [x]** and by 0, respectively. The approach is used for checking partial preorder by condition (9), equivalence, computation of $\oplus$, binary relation Δ, $\mathbf{m}(\alpha)$, and $\mathbf{M}(\alpha)$.

The second approach is based on the fact that, in general, non-smooth functions can be considered as smooth in different regions. Then one can apply the methods developed for the smooth case in these regions and if in all the regions a property is true, it is true also for a given non-smooth function. For example, let $\alpha = x$ and $\beta = |x|$. Then β is smooth in $(-\infty, 0)$ and $[0, \infty)$. In the region $(-\infty, 0)$ $\beta = -x$ and $\beta \leqslant \alpha$, $\alpha \leqslant \beta$ is true. The same is true for the region $[0, \infty)$ and thus $\alpha \cong \beta$.

6. APPLICATIONS

In this section we demonstrate the application of functions' algebra to nonlinear control problems using the developed software. In the first example the accessibility condition from Theorem 6 will be checked.

Example 13. Consider the non-smooth system

$$x_1^+ = x_2 u, x_2^+ = x_1 \operatorname{sign} x_3, x_3^+ = u. \tag{21}$$

First, define the system

```
In[15]:= f = {x2 u, x1 Sign[x3], u}; Xt = {x1, x2, x3}; Ut = {u};
         steq = StateSpace[f, Xt, Ut, t, Shift];
```

Checking accessibility requires finding δ^1

```
In[16]:= δ1 = Delta1[steq]
Out[16]= {x1/x3, x2}
```

and computing the sequence δ^i, $i > 1$ by Algorithm 1, implemented as the function **MinFInvariant**:

```
In[17]:= δk = MinFInvariant[δ1, steq, All]
Out[17]= {{x1/x3, x2}, {x1/x3}, Const, Const}
```

Since the last element of the sequence is constant, the system is, according to Theorem 6, accessible. The simplest way to check accessibility (by Algorithm 1) is to use the command

```
In[18]:= Accessibility[steq, Method → Lattice]
Out[18]= True
```

In order to check the static state feedback linearizability, it is again necessary, according to Theorem 7, to compute the sequence δ^i, $i > 1$. As found in the previous example, for system (21), δ^1 and δ^2 are non-constant functions while δ^3 is a constant function; thus system (21) is feedback linearizable. The new state coordinates, allowing one to find the linearized system equations, may be obtained, by Theorem 7, as follows: the first coordinate $z_1 = \delta^2$; to get the second coordinate, the operator $\mathbf{M}$ has to be applied to δ^2, $z_2 = \mathbf{M}(\delta^2)$, and the third coordinate $z_3 = \mathbf{M}^2(\delta^2)$. The new coordinates are

```
In[19]:= {z1 = δk[[2]], z2 = OperatorCapitalM[z1, steq],
         z3 = OperatorCapitalM[z2, steq]}
Out[19]= {x1/x3, x2, x1 Sign[x2]}
```

The function **Linearization** first checks if the system is linearizable; in the case of an affirmative answer, it finds the linearized equations, state coordinate transformation, and static state feedback[10]:

```
In[20]:= BookForm[Linearization[steq, z#[t]&, v[t], Method → Lattice]]

Out[20]= {{z1+ = z2, z2+ = z3, z3+ = v},
         {z1 = x1/x3, z2 = x2, z3 = x1 Sign[x2], v = z2 u Sign[x3]}}
```

The solution of the DDDPM problem has been implemented as the function **DisturbanceDecoupling**. The detailed algorithm is given in Appendix 1.

Example 14. Consider the system

$$x_1^+ = u + x_1(1+x_2),\ x_2^+ = x_1 + x_4,\ x_3^+ = w + x_3 + x_1(x_2+x_3),$$
$$x_4^+ = x_3 - x_4,\ y_1 = x_1,\ y_2 = x_4,\ y_1^* = x_1.$$

After defining the system by

```
In[21]:= f = {u + x1 (1 + x2), x1 + x4, w + x3 + x1 (x2 + x3), x3 - x4};
         Xt = {x1, x2, x3, x4};
```

[10] The function **BookForm** prints the state equations in the form (1). For the sake of compactness, the equations are given in rows, not in a column.

```
        Yt1 = {y₁, y₂}; h1 = {x₁, x₄}; Yt2 = {y₁*}; h2 = {x₁};
    steq = StateSpace[f, Xt, {{u}, {w}}, t, {h1, h2}, {Yt1, Yt2}, Shift];
```

we can find the decoupled system as follows:

```
In[22]:= BookForm[DisturbanceDecoupling[steq,
           z#[t]&, v#[t]&, Method → Lattice]]

Out[22]= {{z₁⁺ = v₁ + y₁, z₂⁺ = y₁ + y₂, u = v₁ − y₁ z₂},
          {z₁ = x₁, z₂ = x₂}}
```

7. DISCUSSION AND CONCLUSIONS

This paper documents an ongoing effort to implement a mathematical framework, called functions' algebra within *Mathematica*-based symbolic software NLControl. We do hope that the developed software will facilitate further study of various control problems. Moreover, since the functions' algebra approach for nonlinear control systems is developed in full accordance with the algebra of partitions for finite automata [4], it is especially suitable for handling hybrid systems, i.e., systems governed by differential equations whose parameters change due to discrete input or event-driven state transition. The first steps in this direction have been made in [7]. However, recall that in this paper a control system is assumed to be described by difference equations.

The extension of the results for the continuous-time case is not immediate. In general, one cannot find the derivative of a non-smooth function, while forward-shifting of a non-smooth function is not a problem; also, the integration of a vector function is not straightforward and requires the assumption of differentiability of vector functions. In the continuous-time case there is no general formula/algorithm to compute $\mathbf{m}(\alpha)$, and as a consequence, also the sequence $\delta^i, i \geqslant 0$. Moreover, unlike the discrete-time case, the inequality $\delta^{k-1} \geqslant \mathbf{m}(\delta^{k-2})$ does not yield the inequality $\mathbf{M}(\delta^{k-1}) \geqslant \mathbf{M}(\mathbf{m}(\delta^{k-2}))$ on which the proof of Theorem 7 relies.

Finally, a few problems are left for the future study. First, some functions can be improved, most importantly, by finding the simplest representative of the equivalence class and computation of the operator $\mathbf{M}$. Second, several algorithms/programs, described in Section 3, require at some point going to the level of one-forms. This concerns, in particular, checking partial order by condition (9), equivalence, computation of $\oplus$, binary relation Δ, $\mathbf{M}(\alpha)$, and $\mathbf{M}(\alpha)$. If the non-smooth functions are involved, these algorithms/programs require splitting the domain into the regions where the functions are smooth. Currently no software exists for the dividing. Third, currently the approach based on strong partial relation is not supported. Here, again, splitting the domain into suitable regions is necessary.

ACKNOWLEDGEMENTS

The work of A. Kaldmäe, V. Kaparin, Ü. Kotta, and M. Tõnso was supported by the Estonian Research Council, personal research funding grant PUT481. The work of A. Ye. Shumsky and A. N. Zhirabok was supported by a grant of Russian Scientific Foundation, project 16-19-00046. We thank B.P. Podkopaev and the anonymous referee for constructive reviews on the manuscript. The publication costs of this article were covered by the Estonian Academy of Sciences.

APPENDIX 1

DISTURBANCE DECOUPLING ALGORITHM

The function Disturbance Decoupling is based on Algorithm 2 in [6] and follows the steps below.

Given system (6) in the form

system = StateSpace[f, x, {u, w}, t, {h, h*}, {y, y*}, Shift],

state variable of the compensator z, and new input variable v, the algorithm below finds compensator (7) and the relation $z = \alpha(x)$.

Step 1.

1. $\alpha^0 =$ Alpha0[system]
2. $\alpha =$ MinHFInvariant[α^0, system]
3. Define $z = \alpha(x)$ as new variables and construct the system $z^+ = F(z, y, u)$.
4. Check whether $\alpha \leqslant h_*$ by PartialPreOrder[α, h*, x]
   If True, then define h_{**} such that $y_* = h_{**}(z)$. If False, then STOP, the DDDPM is not solvable.
5. Collect the components of z, the function h_{**} depends on, into Z_* by
   $Z_* =$ Cases[h**, _[t], −1].

Step 2.

1. Split the vector y into two disjoint subvectors y_g and y_b. The element y_i of y belongs to y_g if PartialPreOrder[α, h$_i$, x] gives True or
   Complement[Cases[{F}, _[t], −1], Complement[y, y$_i$], u, z] is an empty list.
   All other elements of y belong to y_b: $y_b =$ Complement[y, y$_g$].
2. For $j = 1, \ldots, \dim z$ check whether F_j depends on y_b.
   If Complement[Cases[{F$_j$}, _[t], −1], z, u, y$_g$] === { } gives True, then F_j does not depend on y_b; increase j. Otherwise, check whether F_j depends also on u.
   If Complement[Cases[{F$_j$}, _[t], −1], z, y] === { } gives True, then F_j does not depend on u; add z_j to the set Z_b. Otherwise, increase j.

Step 3.

Until $Z_b = \emptyset$ or remains unchanged, proceed as follows:

1. Let $Z_{vb} = \emptyset$.
2. For each $z_j \in Z_b$ find the function F_i depending on z_j by checking whether
   Complement[Cases[{F$_i$}, _[t], −1], Complement[z, z$_j$], u, y] === { }
   gives False (in which case F_i depends on z_j).
3. Check whether F_i depends on u by
   Complement[Cases[{F$_i$}, _[t], −1], z, y] === { }.
   If True, then add z_i to Z_{vb}; if False, then add z_j to y_b.
4. If $Z_{vb} \cap Z_* \neq \emptyset$, then STOP (no solution), otherwise set $Z_b = Z_{vb}$.

Step 4.

1. Collect all F_i's, which depend on elements from y_b and denote them by F^1. Thus, a function F_i belongs to F^1 if
   Complement[Cases[{F$_i$}, _[t], −1], Complement[z, y$_b$], u, y$_g$] === { }
   gives False.

2. Collect all elements F_i^1 of F^1, which depend on u and denote them by F^2. Thus, a function F_i^1 belongs to F^2 if
   Complement[Cases[{$\mathsf{F_i^1}$}, _[t], −1],z,y] === { } gives False.
3. Construct the system of equations
   equ = Table[$\mathsf{F_i^2}$ == $\mathsf{v_i}$,{i,Length[$\mathsf{F_i^2}$]}], if $\dim F^2 < \dim u$ or
   equ = Table[$\mathsf{F_i^2}$ == $\mathsf{v_i}$,{i,Length[u]}], if $\dim F^2 \geqslant \dim u$.
4. Define
   u^1 = DeleteDuplicates[Complement[Cases[{$\mathsf{F^2}$}, _[t], −1],y,z,v]]
5. Solve the equations equ in variables u^1
   solut = Solve[equ,$\mathsf{u^1}$][[1]]
6. Define
   u^2 = DeleteDuplicates[Complement[Cases[$\mathsf{u^1}$/. solut, _[t], −1],y,z,v]]

Step 5.

1. Substitute u^1 by solut in F.
2. Compute ξ by
   ξ = MaxHFInvariant[h∗∗,StateSpace[F,z,{u,v,y},t,{ },{ },Shift]][[−1]]
3. For all i, check whether ξ_i^+ depends on y_b. If not for all i, then END. If yes, then check whether ξ_i^+ depends on u. If not, STOP (no solution), otherwise, let $u = u^2$ and return to Step 4.
   If the algorithm finds the solution, it gives
   {StateSpace[F,z,v,t,u/. solut,u,Shift], $z == \alpha(x)$}

REFERENCES

1. Aranda-Bricaire, E., Kotta, Ü., and Moog, C. H. Linearization of discrete-time systems. *SIAM J. Contr. Optim.*, 1996, **34**(6), 1999–2023.
2. Belikov, J., Kaparin V., Kotta, Ü., and Tõnso, M. *NLControl website*. Online, `www.nlcontrol.ioc.ee`, 2016 (accessed 20 June 2016).
3. Conte, G., Moog, C. H., and Perdon, A. M. *Algebraic Methods for Nonlinear Control Systems*. Springer, London, 2007.
4. Hartmanis, J. and Stearns, R. E. *The Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, New York, 1966.
5. Isidori, A. *Nonlinear Control Systems*. Springer, London, 1995.
6. Kaldmäe, A., Kotta, Ü., Shumsky, A. Ye., and Zhirabok, A. N. Measurement feedback disturbance decoupling in discrete-time nonlinear systems. *Automatica*, 2013, **49**(9), 2887–2891.
7. Kaldmäe, A., Kotta, Ü., Shumsky, A. Ye., and Zhirabok, A. N. Disturbance decoupling in nonlinear hybrid systems. In *12th IEEE International Conference on Control & Automation (ICCA), Kathmandu, Nepal*. 2016, 86–91.
8. Kaparin, V., Kotta, Ü., Shumsky, A. Ye., Tõnso, M., and Zhirabok, A. N. Implementation of the tools of functions' algebra: First steps. In *Proceedings of MATHMOD 2012 – 7th Vienna International Conference on Mathematical Modelling, Vienna, Austria*. 2012, 1231–1236.
9. Kotta, Ü. and Tõnso, M. Linear algebraic tools for discrete-time nonlinear control systems with Mathematica. In *Nonlinear and Adaptive Control, NCN4 2001*. Lecture Notes in Control and Information Sciences, 2003, 281, 195–205.
10. Kotta, Ü., Tõnso, M., Belikov, J., Kaldmäe, A., Kaparin, V., Shumsky, A. Ye., and Zhirabok, A. N. A symbolic software package for nonlinear control systems, In *Proceedings of the 2013 International Conference on Process Control (PC), Štrbské Pleso, Slovakia*. 2013, 101–106.
11. Kotta, Ü., Tõnso, M., Shumsky, A. Ye., and Zhirabok, A. N. Feedback linearization and lattice theory. *Syst. & Contr. Lett.*, 2013, **62**(3), 248–255.
12. Nijmeijer, H. and van der Schaft, A. J. *Nonlinear Dynamical Control Systems*. Springer, New York, 1990.
13. Zhirabok, A. N. and Shumsky, A. Ye. *The Algebraic Methods for Analysis of Nonlinear Dynamic Systems*. Dalnauka, Vladivostok, 2008 (in Russian).

Funktsioonide algebra mittelineaarsete juhtimisüsteemide uurimisel: arvutuslikud aspektid ja tarkvara

Juri Belikov, Arvo Kaldmäe, Vadim Kaparin, Ülle Kotta, Alexey Ye. Shumsky, Maris Tõnso ja Alexey Zhirabok

On kirjeldatud *Mathematica* keskkonnas väljatöötatud tarkvara (ja selle aluseks olevaid algoritme), mis realiseerib matemaatilisel lähenemisel, mida nimetatakse funktsioonide algebraks, põhinevaid meetodeid mittelineaarsete juhtimissüsteemide valdkonnas olevate probleemide lahendamiseks. Antud lähenemise eelis tuntud diferentsiaalgeomeetria ja diferentsiaalsetel 1-vormidel põhinevate meetoditega võrreldes on rakendatavus ka mittesiledatele süsteemidele. Puuduseks on arvutuste suurem keerukus, kuna lähenemine opereerib vahetult süsteemi kirjeldavate funktsioonidega ja mitte nende diferentsiaalidega, mis lihtsustavad (lineariseerivad) arvutusi.

Tarkvaraliselt on realiseeritud funktsioonide algebra põhioperatsioonid ja seosed, nagu osaline järjestus, ekvivalents, liitmine ning korrutamine, aga ka lihtsaima esindaja leidmine ekvivalentsiklassis. Teine suur rühm *Mathematica* funktsioone on seotud juhtimissüsteemiga defineeritud binaarse suhte ja operaatoritega **m** ning **M**, aga ka teatud invariantsete funktsioonide jada leidmisega süsteemi võrrandite põhjal. Kolmas hulk funktsioone lahendab hulga juhtimisülesandeid diskreetsete mittelineaarsete juhtimissüsteemide jaoks, nagu juhitavuse kontroll, staatilise olekutagasisidega olekuvõrrandite lineariseerimine ja häiringute dekomponeerimine väljundtagasisidega. Artiklis toodud näited demonstreerivad tarkvara kasutamist.