An algorithm for determining a class of invariants in quasi-polynomial systems*

Barna Pongrácz[1,2], Gábor Szederkényi[1], Katalin M. Hangos[1]

[1]Process Control Research Group, Systems and Control Laboratory
Computer and Automation Research Institute,
Hungarian Academy of Sciences
H-1518, P.O. Box 63, Budapest, Hungary
Tel: +36 1 279 6000
Fax: +36 1 466 7503
e-mail: pongracz@scl.sztaki.hu, szeder@sztaki.hu, hangos@scl.sztaki.hu
[2]Department of Computer Science, University of Veszprém
H-8200, Egyetem u. 10, Veszprém, Hungary

*This document is the personal version of the article published in: B. Pongrácz, G. Szederkényi and K.M. Hangos. An algorithm for determining a class of invariants in quasi-polynomial systems. *Computer Physics Communications*, 175 (3): 204-211, 2006. doi:10.1016/j.cpc.2006.03.003

Abstract

In this paper an algorithm is proposed for the retrieval of a wide class of invariants in quasi-polynomial systems. The invariance properties of the algorithm under different transformations are discussed. The application of the algorithm is illustrated on physical and numerical examples. The algorithm has been implemented in the MATLAB computing environment.

PACS codes: 05.45.-a, 02.30.Ik, 89.20.Ff
Keywords: Quasi-polynomial systems, first integrals

1 Introduction

Finding constants of motion has been an important and intensively studied area of the analysis of dynamical systems for a long time. If the given dynamical system is not integrable, then its first integrals (if they exist) give us very useful information about the properties of the solutions and about possibly physically meaningful conserved quantities. Several different approaches have been proposed in the literature for the determination of invariants under various conditions (see e.g. [3, 15, 19]). Furthermore, first integrals play a great role in modern systems and control theory e.g. in the field of canonical representations, controllability and observability analysis [14] and stabilization of nonlinear systems [7, 16].

The class of quasi-polynomial (QP) systems has gained a significant interest in the modelling of nonlinear dynamical systems since the majority of smooth nonlinear systems occurring in practice can be algorithmically transformed to QP form [4, 12, 13]. Recent research indicates that QP system-representation can be used as a very useful tool in nonlinear systems and control theory [17, 20, 21], too.

The theoretical background of the existence of quasi-polynomial invariants is well-founded. In [8] algebraic tools are applied to find semi-invariants and invariants in quasi-polynomial systems. In [9] it is shown that the existence of polynomial-type semi-invariants in the corresponding Lotka-Volterra systems is a necessary condition for the existence of quasi-polynomial invariants. A general method is given in the same paper for the symbolical checking of this necessary condition with numerous valuable examples. Moreover, a computer-algebraic software package called QPSI has been implemented for the determination of quasi-polynomial invariants and the corresponding model parameter relations [10].

The purpose of this paper is to propose a numerically effective algorithm for the retrieval of a frequent class of quasi-polynomial invariants. The main differences between the approach of QPSI and our method are the following. Firstly, the classes of invariants that can be found by QPSI and our algorithm are different. Our method is able to retrieve only those invariants that can be written as an explicit function of a power of a differential variable. This is a narrower class of first integrals than QPSI can handle, but this restriction is not so severe in practice (see section 2.2). Secondly, our method does not use the Lotka-Volterra representation of QP systems, but it searches for invariants directly in the original QP system model. Additionally, our algorithm is not based on finding semi-invariants. Thirdly, although our algorithm itself can handle the model parameters symbolically, the implementation has been done in a numerical computational environment (MATLAB) while QPSI works in a symbolic software environment (Maple). It follows from the above mentioned differences that our algorithm works effectively even in those cases when the number of monomials is high (see the examples in section 5).

This paper is organized as follows. Section 2 contains the basic notions that are needed to derive the main results. The main contribution of the paper can be found in section 3 where the algorithm for the retrieval of invariants is described. In section 4 the invariance properties of this algorithm are discussed. Section 5 contains four physical and numerical examples for the illustration of the proposed algorithm.

2 Basic notions

2.1 Quasi-polynomial systems

Let us denote the element of an arbitrary matrix W with row index i and column index j by $W_{i,j}$. Furthermore, let the i-th row and j-th column of W denoted by $W_{i,\cdot}$ and $W_{\cdot,j}$ respectively. Quasi-polynomial systems are systems of ODEs. An $(n+1)$ dimensional QP-ODE system can be represented in the following general form:

$$\dot{x}_i = x_i \left(\lambda_i + \sum_{j=1}^{m-1} \bar{A}_{i,j} U_j \right) \quad , \quad U_j = \prod_{k=1}^{n+1} x_k^{\bar{B}_{j,k}}, \qquad i = 1, \ldots, n+1 \tag{1}$$

where $\bar{A} \in \mathbb{R}^{(n+1)\times(m-1)}$, $\bar{B} \in \mathbb{R}^{(m-1)\times(n+1)}$, $\lambda_i \in \mathbb{R}$, $i = 1, \ldots, n+1$. Furthermore, $\lambda = [\lambda_1 \ \ldots \ \lambda_{n+1}]^T$. The product terms U_j, $j = 1, \ldots, m-1$ are called the quasi-monomials (or monomials) of the system. Without the loss of generality we can assume that $m - 1 \geq n + 1$, and that the matrices $\bar{A}$ and $\bar{B}$ are of full rank i.e. $\text{Rank}(\bar{A}) = \text{Rank}(\bar{B}) = n + 1$ [13].

If $\lambda_i \neq 0$ for some i, then it is useful to introduce a so-called unit monomial $U_m = \prod_{k=1}^{n+1} x_k^0 = 1$. This way, the general equations (1) can be written in a *homogeneous* form (see e.g. [5] or [11]) as

$$\dot{x}_i = x_i \left(\sum_{j=1}^{m} A_{i,j} U_j \right) \quad , \quad U_j = \prod_{k=1}^{n+1} x_k^{B_{j,k}}, \qquad i = 1, \ldots, n+1 \tag{2}$$

where the matrices $A \in \mathbb{R}^{(n+1)\times m}$ and $B \in \mathbb{R}^{m\times(n+1)}$ are the following:

$$\begin{aligned} A_{i,j} &= \bar{A}_{i,j}, \quad i = 1, \ldots, n+1; \ j = 1, \ldots, m-1 \\ A_{i,m} &= \lambda_i, \quad i = 1, \ldots, n+1 \end{aligned}$$

and

$$\begin{aligned} B_{i,j} &= \bar{B}_{i,j}, \quad i = 1, \ldots, m-1; \ j = 1, \ldots, n+1 \\ B_{m,j} &= 0, \quad j = 1, \ldots, n+1 \end{aligned}$$

i.e. λ is inserted as a column vector after the last column of $\bar{A}$, and a zero row vector is inserted after the last row of $\bar{B}$. The facts that $\bar{A}$ and $\bar{B}$ contain

$n+1$ linearly independent rows and columns, respectively, and the row and column ranks of a matrix are equal, imply that the rank of both A and B remains $n+1$. Then, it follows from the relation $n+1 < m$ that A and B are also of full rank, which is a necessary condition for the applicability of the proposed algorithm.

2.2 The examined class of invariants

A function $I : \mathbb{R}^{n+1} \mapsto \mathbb{R}$ is called an invariant of (2) if

$$\frac{d}{dt} I = \frac{\partial I}{\partial x} \cdot \dot{x} = 0. \tag{3}$$

We consider quasi-polynomial invariants in (2) that can be written in the following special form:

$$I = F(x) - x_i^{\frac{1}{\beta}}, \quad \beta \in \mathbb{R} \tag{4}$$

where

$$F(x) = \sum_{k=1}^{p} c_k \prod_{j=1, j \neq i}^{n+1} x_j^{\alpha_{kj}}, \quad c_k, \alpha_{kj} \in \mathbb{R} \tag{5}$$

It's clear that (4) can be rewritten as

$$x_i^{\frac{1}{\beta}} = F(x) + c_0, \quad c_0 \in \mathbb{R} \tag{6}$$

This is a narrower class of invariants than the one examined in [8] since it contains those first integrals from where at least one of the variables can be expressed explicitly. However, many types of first integrals (e.g. conserved mechanical, thermodynamical or electrical energy) in physical system models belong to this class.

3 The algorithm for retrieving invariants

In the following, an algorithm will be presented which is capable to retrieve single invariants described in section 2.2. With a slight modification, another algorithm is given which is able to retrieve multiple first integrals from a QP-ODE model. Then the MATLAB implementation and computational properties of these algorithms are discussed.

3.1 The underlying principle of the algorithm

Consider a set of $(n+1)$ QP differential equations in the *homogeneous* form of (2). Let us assume without restriction of generality that $i = n+1$ in (6) (because the QP form of the equations is preserved under permutation of the differential variables) i.e. the following algebraic dependence is present in (2)

$$x_{n+1}^{\frac{1}{\beta}} = c_0 + \sum_{\ell=1}^{L} c_\ell V_\ell \tag{7}$$

where $\beta, c_\ell \in \mathbb{R},\ \ell = 0, \ldots, L,\ \beta \neq 0$, and

$$V_\ell = \prod_{k=1}^{n} x_k^{\alpha_{\ell k}}, \quad \alpha_{\ell k} \in \mathbb{R},\ \ell = 1, \ldots, L,\ k = 1, \ldots, n \tag{8}$$

It is clear that (7) is equivalent to the existence of a first integral of the form (4)-(5).

Taking the time derivative of (7) we obtain

$$\dot{x}_{n+1} = \beta \left(c_0 + \sum_{\ell=1}^{L} c_\ell V_\ell \right)^{\beta-1} \cdot \sum_{\ell=1}^{L} c_\ell \dot{V}_\ell \tag{9}$$

Using (7) and the fact that the monomials V_ℓ, $\ell = 1, \ldots, L$ do not depend on x_{n+1} we can further write

$$\dot{x}_{n+1} = \beta x_{n+1}^{\frac{\beta-1}{\beta}} \sum_{\ell=1}^{L} c_\ell \cdot \sum_{i=1}^{n} \frac{\partial V_\ell}{\partial x_i} \dot{x}_i \tag{10}$$

Finally, we can rewrite (10) to the standard QP form as

$$\dot{x}_{n+1} = x_{n+1} \left(\sum_{\ell=1}^{L} \sum_{i=1}^{n} \beta \cdot c_\ell \cdot \alpha_{\ell i} \cdot V_\ell \cdot x_{n+1}^{-\frac{1}{\beta}} \sum_{j=1}^{m} A_{i,j} U_j \right) \tag{11}$$

It is easy to see that the monomials in (11) (denoted by $R_{\ell j}$) are the following

$$R_{\ell j} = V_\ell \cdot U_j \cdot x_{n+1}^{-\frac{1}{\beta}} = x_1^{\alpha_{\ell 1} + B_{j1}} \cdot x_2^{\alpha_{\ell 2} + B_{j2}} \cdot \cdots \cdot x_n^{\alpha_{\ell n} + B_{jn}} \cdot x_{n+1}^{-\frac{1}{\beta}} \tag{12}$$

$$j = 1, \ldots, m, \quad \ell = 1, \ldots, L \tag{13}$$

while the coefficients of the monomials are

$$\gamma_{\ell j} = \sum_{i=1}^{n} \beta c_\ell \alpha_{\ell i} A_{i,j} \tag{14}$$

$$i = 1, \ldots, n, \quad j = 1, \ldots, m, \quad \ell = 1, \ldots, L \tag{15}$$

where the subscript i refers to that the partial differentiation in (10) has been performed by x_i.

Now, the aim of our algorithm is to determine β, the coefficients c_ℓ and the exponents $\alpha_{\ell i}$, $\ell = 1, \ldots, L$, $i = 1, \ldots n$ in (7)–(8) using the special form of the equation (11) and that of the monomials in (12).

3.2 The basic algorithm for retrieving single invariants

The *input required by the algorithm* consists of the matrices A and B of the QP model in its *homogeneous form* defined in (2).

The *operational condition of the algorithm* is that, consistently to our preliminary assumptions, matrices A and B are of full rank.

Without the loss of generality we can assume that the explicit variable of the possible first integral is the last differential variable x_{n+1}. By a simple permutation of variables, each variable can be checked whether it is the explicit variable of a first integral.

1. *Determination of the monomial candidates*
   To find a first integral in the form (7), one has to use the relationship (12) defined between the monomials $U_j, j = 1, \ldots, m$ of the original differential equations and the monomials $R_{\ell j}, \ell = 1, \ldots, L, j = 1, \ldots, m$ of the ODE for the algebraically dependent variable x_{n+1}.

   The first step is dedicated to collect these two groups of monomials, and then to determine the monomial candidates of the first integral using (12). Since the exponents of the j-th monomial of a QP-ODE are given as the j-th row vector of matrix B, the first thing to do is to gather the exponents of those monomials that occur in the first n differential equations and construct the matrix $B^{(U)}$ from them. Let us denote the matrix created form A by deleting its $(n+1)$-th row by A^*. Now construct $B^{(U)}$ in the following way:

> *Let* $B^{(U)} = B$
> *Mark those rows* $B^{(U)}_{j,\cdot}$, $j = 1, \ldots, m$, *for which* $A^*_{\cdot,j} = 0$
> *Delete the marked rows from* $B^{(U)}$

Similarly, collect the row vectors containing the exponents of monomials of the ODE for x_{n+1} to $B^{(R)}$:

> *Let* $B^{(R)} = B$
> *Mark those rows* $B^{(R)}_{j,\cdot}$, $j = 1, \ldots, m$, *for that* $A_{(n+1),j} = 0$
> *Delete the marked rows from* $B^{(R)}$

As a result, $B^{(U)} \in \mathbb{R}^{m_U \times (n+1)}$, $B^{(R)} \in \mathbb{R}^{m_R \times (n+1)}$, where m_U, m_R denote the number of monomials in the first n, and in the $(n+1)$-th differential equations, respectively. Note that there may be monomials (as row vectors) that appear in both $B^{(U)}$ and $B^{(R)}$.

As (12) shows, the exponents of the monomials U_j, $j = 1, \ldots, m$ in the original differential equations and of the monomials $V_\ell, \ell = 1, \ldots, L$ in the algebraic equation are added up in the resulted monomials $R_{\ell j}$. This allows us to determine V_ℓ by simply *dividing* $R_{\ell j}$ by U_j for some j. This operation is equivalent to subtracting each exponent row vector corresponding to the monomials U_j (stored as row vectors of $B^{(U)}$) from the row vectors determining $R_{\ell j}$ (stored as row vectors of $B^{(R)}$).

Therefore the next step is that the algorithm determines the exponent row vectors by subtracting each row of $B^{(U)}$ from each row of $B^{(R)}$, and construct the matrix $B^{(V)}$ made of the resulted row vectors:

$$B^{(V)} \in \mathbb{R}^{(m_U \cdot m_R) \times (n+1)} \ : \ B^{(V)}_{k,\cdot} = B^{(R)}_{j,\cdot} - B^{(U)}_{i,\cdot} \ , \quad k = (j-1) \times m_U + i$$

Finally, make sure that each monomial candidate is coded only once in $B^{(V)}$:

> *Delete repeated rows from* $B^{(V)}$ *so that all rows are different*

As a result, $B^{(V)} \in \mathbb{R}^{m_V \times (n+1)}$ contains all the monomial candidates of the first integral, where $m_V \leq m_U \cdot m_R$ denotes the number of these monomial candidates. However, taking into account all possible monomial candidates may cause a huge redundancy but this guarantees that the exponent vectors of *all monomials of the first integral are contained* in $B^{(V)}$.

2. *Determination of* β

   To have a QP-type first integral from which x_{n+1} can be given explicitly, *the exponents of* x_{n+1} *in all of its monomials have to be identical.* This step classifies the exponent row vectors of the monomial candidates of the first integral by their last element.

   Compute how many different last elements of the row vectors of $B^{(V)}$ have and denote this number by S. Now make S different sets $\mathcal{B}_k,\ k = 1, \ldots, S$ and collect all the row vectors of $B^{(V)}$ having identical last elements into the same sets, while row vectors with different last elements into different sets. The result is a system of sets, where *the elements of each set are exponent row vectors belonging to the same* β.

   Now set the value of k to $k = 1$.

3. *Determination of the coefficients*

   The last step to be performed is to search for a first integral with monomial candidates belonging to the same $\mathcal{B}_k$. Since the exponents of the monomial candidates are already given, only their coefficients have to be determined. If these coefficients exist, the first integral exists for the current β, and it is completely determined by the algorithm.

   Denote the number of elements of $\mathcal{B}_k$ by L, the candidate for being the number of quasi-monomials in (7). Then the first integral candidate is given by the monomials described by the elements of $\mathcal{B}_k$ with unknown coefficients $c_1, \ldots, c_L$. Perform time-differentiation by simply applying (11) to it, with monomials and coefficients described in (12) and (14), respectively. Then match the monomials of this time-derivative and the monomials of the $(n+1)$-th differential equation, and determine the coefficients $\gamma_{\ell j}$ $, \ell = 1, \ldots, L,\ j = 1 \ldots, m$ therefrom. Then try to solve the linear set of equations (14) for $c_1, \ldots, c_L$.

   Three cases are possible:

(a) If (14) cannot be solved and $k < S$, increase k by one and jump to Step 3.

(b) If (14) cannot be solved and $k = S$ the algorithm stops without finding a first integral.

(c) If (14) can be solved, then the first integral is successfully determined and the algorithm stops.

This algorithm is capable of finding QP type first integrals which are explicit in (at least) one of their variables, moreover it operates without any heuristic steps. The properties of the algorithm are thoroughly discussed in section 4.

3.3 Retrieval of multiple first integrals

The multiple retrieval algorithm comes from the basic algorithm by two simple modifications:

1. The first modification is on Step 3.(c), where the algorithm *does not stop* but stores the first integral found, increases s by one, and jumps back to Step 3.

2. Similarly to the basic algorithm, the multiple retrieval algorithm can find first integrals which are explicit in x_{n+1}. The second modification is the application of an outer loop performed $(n+1)$ times applying a variable index swapping in each steps: in the k-th swapping

   $$x_k^{new} = x_{n+1}, \quad x_{n+1}^{new} = x_k, \quad x_i^{new} = x_i, \quad i = 1, \ldots, n, \quad i \neq k$$

   where the superscript *'new'* refers to the variables with swapped indices. With this modification, the multiple retrieval algorithm can find first integrals which are explicit in at least one of their variables.

Care has to be taken to first integrals that are explicit in more than one of their variables, because the multiple retrieval algorithm finds these algebraic dependencies several times. Thus, after running this algorithm, the linear independence of invariants found has to be checked.

3.4 Implementation and computational properties of the algorithms

The basic and multiple retrieval algorithms are implemented in the MATLAB computation environment [1]. The functions

```
[TEXTFORM,C,EXPONENTS,MESSAGE]=ALG_BASIC(A,B)
[TEXTFORM,WHICHONE,C,EXPONENTS,MESSAGE]=ALG_MULTIPLE(A,B)
```

call the basic and multiple retrieval algorithms, respectively. The inputs A and B are the matrices of the QP-ODE model in its homogeneous form (2), while the output `TEXTFORM` gives the explicit form of invariant(s) found in a readable (text) format, while the outputs `WHICHONE`, `C` and `EXPONENTS` give the index of the explicit variable, the coefficients and the exponents of the first integral, respectively.

Both algorithms are of polynomial complexity. Recall that the number of the sets $\mathcal{B}_k$ was denoted by S in step 3 of section 3.2. Furthermore, let u denote the maximal element size of $\mathcal{B}_k$, $k = 1, \ldots, S$. The retrieval of a *single* invariant with the basic algorithm is of order

$$\Theta\Big(S \times u^3 \times m^3\Big)$$

where m denotes the number of monomials of the QP-ODE. The retrieval of *multiple* first integrals with the multiple retrieval algorithm is of order

$$\Theta\Big(S \times u^3 \times m^3 \times n\Big)$$

A rough upper estimation of these orders are $\Theta\big(m^7\big)$ and $\Theta\big(m^7 \times n\big)$, respectively, where m is the number of monomials, while $n + 1$ is the number of equations in (2).

Although MATLAB gives numerical solutions, there is a possibility to find first integrals even in parametric form (see Example 2 - Rikitake system) because for solving sets of linear equations, the basic and the multiple retrieval algorithms use the '`linsolve`' command of the built-in Maple kernel of MATLAB Symbolic Math Toolbox utilizing its efficiency [2].

4 Algebraic properties of the algorithm

Being state-space models, QP-ODEs are inherently not unique in the sense that there are different QP-ODE models describing the same physical system: there are coordinates transformations of the variables that transform the model to an equivalent QP-ODE. Moreover there might be another transformations that give a different, but equivalent QP-ODE model.

This section concerns with the invariance of the retrieval process under two of these transformations performed on the QP-ODE model: under (nonlinear) quasi-monomial (QM) state transformations, and under algebraic equivalence transformations.

4.1 The effect of quasi-monomial transformations

Consider the general form of invertible quasi-monomial transformations (QMT) [13]:

$$\hat{x}_i = \prod_{j=1}^{n+1} x_i^{C_{i,j}} \quad , \quad i = 1, \dots, n+1$$

where $C \in \mathbb{R}^{(n+1)\times(n+1)}$ is a invertible quadratic matrix, $x_i,\ i = 1, \dots, n+1$ denotes the original and $\widehat{x}_i, i = 1, \dots, n+1$ the transformed coordinates. The inverse of this transformation is also a QMT characterized by C^{-1} [6]. It is shown in [9] that a QP invariant of the form

$$I = \sum_{i=1}^{N} F_i \prod_{k=1}^{n} x_k^{E_{i,k}} \tag{16}$$

has the following form in the transformed coordinates:

$$I = \sum_{i=1}^{N} F_i \prod_{k=1}^{n} \hat{x}_k^{[EC^{-1}]_{i,k}} \tag{17}$$

which means that (17) has the same quasi-monomial terms as (16). It is easy to see from (4) and (17) that the explicitness of $x_i^{\frac{1}{\beta}}$ is generally not preserved under QMTs. Assuming that the index of the explicit variable is $i = n+1$ in (4), it is clear that a QMT with the following special C matrix preserves

x_{n+1} as the explicit variable:

$$C = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline 0 & C_{22} \end{array} \right]$$

where $C_{11} \in \mathbb{R}^{n\times n},\ C_{12} \in \mathbb{R}^{n\times 1},\ C_{22} \in \mathbb{R},\ 0 \in \mathbb{R}^{1\times n}$.

It can be seen from the description in section 3, that the first step of the algorithm does not use the special explicit structure of the invariants but the second and third steps are completely dependent on this property. However, this dependence can be relaxed a bit on the price of more computation time in the following way. In Step 2, all the exponent row vectors $b^{(V)}$ of the monomial candidates are put to the *same set* $\mathcal{B}_1$ *irrespectively of their last elements.* Then the first integral candidate can be written *in its implicit form* with the monomials described by the exponent rows vectors of $\mathcal{B}_1$, and with unknown coefficients of the monomials. These coefficients can be determined by the time-differentiation of this first integral candidate.

This modification yields to first integrals that may not be given explicitly, only after an appropriate QM state transformation. This extension also generates a significant redundance of monomial candidates possibly causing numerical problems in Step 3 during the determination of the monoms' coefficients of the first integral.

4.2 The effect of algebraic equivalence transformations

It is important to remark that there are cases when the sum of the coefficients in (11) is such that too much information is lost (i.e. too many terms are cancelled) to be able to obtain candidates for the parameters of (7)-(8) by the proposed algorithm. Of course, this problem can be caused by an unlucky equivalent algebraic transformation of the right hand side of the differential equations, as we will see at the end of Example 1 in section 5. However (as it is also shown in Example 1), there is a good hope to finally find the first integral if it is explicit in at least one more variable.

5 Examples

In this section, the operation of the basic algorithm is illustrated on the model of a fed-batch fermentation process, and on the model of a simple electric circuit. The multiple retrieval algorithm is also demonstrated on a Rikitake system and on a computational example. Most of these examples are implemented in MATLAB environment.

5.1 Example 1: A fed-batch fermentation process

In this example we use the model of an isotherm fed-batch fermentation process. Fermenters are used for producing different kinds of biomass (e.g. baker's yeast, antibiotics, etc.) in various branches of industry.

A fed-batch fermentation process with a bi-linear reaction characteristics [18] is used as a case study which is described by the following physical model:

$$\dot{x}_1 = K_r x_1 x_2 - \frac{x_1}{x_3} F \tag{18}$$

$$\dot{x}_2 = -\frac{1}{Y} K_r x_1 x_2 + \frac{S_F - x_2}{x_3} F \tag{19}$$

$$\dot{x}_3 = F \tag{20}$$

The variables of the model and their units in square brackets are

x_1	biomass concentration	[g/l]
x_2	substrate concentration	[g/l]
x_3	volume	[l]
F	feed flow rate	[l/h].

The constant parameters and their typical values are the following

$Y = 0.5$	yield coefficient	
$K_r = 1$	kinetic parameter	[g/l]
$S_F = 10$	influent substrate concentration	[g/l]

The feed flow-rate F is the physical control input variable and will be treated as a constant parameter during the retrieval process. The model (18-20) can be given as a QP-ODE in its homogeneous form (2):

$$\dot{x}_1 = x_1\big(K_r x_2 - F x_3^{-1}\big) \tag{21}$$

$$\dot{x}_2 = x_2\big(-\frac{1}{Y} K_r x_1 + S_F F x_2^{-1} x_3^{-1} - F x_3^{-1}\big) \tag{22}$$

$$\dot{x}_3 = x_3\big(F x_3^{-1}\big) \tag{23}$$

The A and B matrices of the QP model are:

$$A = \begin{bmatrix} K_r & -F & 0 & 0 \\ 0 & -F & -\frac{K_r}{Y} & S_F F \\ 0 & F & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & -1 \end{bmatrix}$$

Apply the basic algorithm to search for a first integral. First set x_1 the variable for which the first integral is searched for. It needs to put x_1 to the 3-rd place, which can be easily done with the following change of coordinates: $x_1^{new} = x_3,\ x_2^{new} = x_2,\ x_3^{new} = x_1$.

Now apply the basic algorithm:

Step 1. The system matrices in the new coordinates are:

$$A^{new} = \begin{bmatrix} 0 & F & 0 & 0 \\ 0 & -F & -\frac{K_r}{Y} & S_F F \\ K_r & -F & 0 & 0 \end{bmatrix}, \quad B^{new} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$

The matrices $B^{(U)}$ and $B^{(R)}$ containing the exponent row vectors of the monomials respectively in the first two, and in the third differential equations are:

$$B^{(U)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \quad B^{(R)} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

Now subtract each row of $B^{(U)}$ from each row of $B^{(R)}$:

$$\begin{aligned}
b_1^{(V)} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} - \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \\
b_2^{(V)} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \\
b_3^{(V)} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} - \begin{bmatrix} -1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \end{bmatrix} \\
b_4^{(V)} &= \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\
b_5^{(V)} &= \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & -1 \end{bmatrix} \\
b_6^{(V)} &= \begin{bmatrix} -1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} -1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}
\end{aligned}$$

Since the last elements of these row vectors should give $-\frac{1}{\beta}$, these elements must be non-zero, meaning that there are only two feasible exponent vectors: $b_2^{(V)}$ and $b_5^{(V)}$.

Step 2. Since the last components of both exponent vectors are identical, (both vectors belong to the same $\beta = 1$) there is only one set $\mathcal{B}_1$ containing these two exponent vectors: $\mathcal{B}_1 = \{b_2, b_5\}$.
Step 3. The parametric first integral is

$$x_3^{new} = c_1 (x_1^{new})^{-1} + c_2 x_2^{new} + c_0 \tag{24}$$

Its time-derivative does not provide solution for $c_1,\ c_2$ because of the phenomena described in section 4.2. However, this can be written in the original coordinates as

$$x_1 = c_1 x_3^{-1} + c_2 x_2 + c_0 \tag{25}$$

By arranging it to an implicit form:

$$x_3 \big(x_1 - c_2 x_2 - c_0 \big) = c_1 \tag{26}$$

and taking its time-derivative gives an equation that from the coefficients c_1 and c_3 can be uniquely determined, and therefore the implicit form of the first integral is retrieved:

$$x_3 \big(\frac{1}{Y} x_1 - x_2 + S_F \big) = \frac{S_F}{c_0} \tag{27}$$

where c_0 comes from the initial conditions of the model.

Note that another algorithm based on the construction of Lie-algebras has been applied to a similar fermenter model with the same structure but slightly different reaction kinetics in [22], giving the same final result. As a comparison, this new method provides a computationally more advantageous way of retrieval mainly because it does not require the analytic solution of partial differential equations which was the case in [22].

Now take a look at the four monomials $U_1, \ldots, U_4$ of the model:

$$U_1 = x_2, \quad U_2 = x_3^{-1}, \quad U_3 = x_1, \quad U_4 = x_2^{-1} x_3^{-1} \tag{28}$$

To see the connection with [8], the monomials of the first integral (27) can be written as a product of a term which is a quasi-monomial function of the monomials, and another term which is a polynomial function of the monomials:

$$U_2^{-2} \Big(\frac{1}{Y} U_2 U_3 - U_1 U_2 + S_F U_2 \Big) = \frac{S_F}{c_0} \tag{29}$$

Observe that the polynomial term is of second order.

This example shows the effect of an algebraic equivalence transformation described in section 4.2. If one chooses x_3 as the explicit variable of the first integral, then the retrieval process is not successful. To understand the exact problem, let us express x_3 form the first integral (27):

$$x_3 = \Big(\frac{c_0}{S_F}\big(\frac{1}{Y}x_1 - x_2 + S_F\big)\Big)^{-1} \tag{30}$$

then take its time-derivative:

$$\dot{x}_3 = -x_3^2\big(-\frac{1}{Y}\dot{x}_1 - \dot{x}_2\big) = x_3\big(\frac{c_0 F}{S_F Y}x_1 + c_0 F - \frac{c_0 F}{S_F}x_2\big) \tag{31}$$

and then compare this to (23). These two differential equations are *equivalent*, since

$$\frac{c_0 F}{S_F Y}x_1 + c_0 F - \frac{c_0 F}{S_F}x_2 = F x_3^{-1} \tag{32}$$

according to (27). This is the case when an algebraic equivalence transformation defined by (32) is applied to the third model equation (31) resulting (23).

5.2 Example 2: A Rikitake system

Consider a Rikitake system [8] with $\alpha = \mu = 0$:

$$\begin{aligned} \dot{x}_1 &= x_1(x_1^{-1}x_2x_3 + \beta x_1^{-1}x_2) && (33)\\ \dot{x}_2 &= x_2(x_1x_2^{-1}x_3 - \beta x_1x_2^{-1}) && (34)\\ \dot{x}_3 &= x_3(x_1x_2x_3^{-1}) && (35) \end{aligned}$$

The application of the basic algorithm gives the following first integral:

$$x_2^2 = c_1x_1^2 - 2\beta(1+c_1)x_3 + (1-c_1)x_3^2 + c_0 \ , \quad c_1 \in \mathbb{R} \tag{36}$$

where the constant c_0 comes from the initial conditions of the model while c_1 is an arbitrary real parameter. This first integral is equivalent with two non-parametric first integrals:

$$\begin{aligned} I_1 &= x_3^2 + 2\beta x_3 - x_1^2\\ I_2 &= x_3^2 - 2\beta x_3 - x_2^2 \end{aligned}$$

We note that the first integral found in [8] is the linear combination $I_1 - I_2$.

The MATLAB implementation of this example uses a constant β value and applies the multiple retrieval algorithm to show that the first integral comes back with all variables chosen as the explicit one.

5.3 Example 3: An electric circuit

Consider the simple electric circuit in Figure 1. consisting of two inductors and a capacitor. Assume that the inductors have linear characteristics with

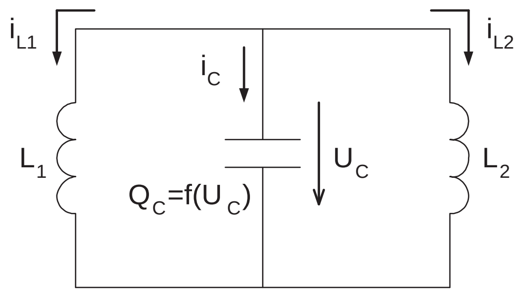

Figure 1: The LC circuit

inductances L_1 and L_2. Denote the currents of the inductors by i_{L1} and i_{L2} respectively. Let the capacitor have a nonlinear QM-type characteristic:

$$Q_C = f(U_C)$$

where Q_C is the charge and U_C is the voltage of the capacitor and f is a QM-type function of U_C. The partial derivative of f is also a QM-type function of U_C:

$$\frac{\partial f}{\partial U_C} = kU_C^{\alpha}\ , \quad \alpha \in \mathbb{N}, k \in \mathbb{R}$$

Denote the current of the capacitor by i_C. The dependence

$$i_C = \frac{dQ_C}{dt} = \frac{\partial Q_C}{\partial U_C} \cdot \frac{dU_C}{dt}$$

gives

$$\frac{dU_C}{dt} = \frac{1}{\frac{\partial f}{\partial U_C}} \cdot i_C$$

Using Kirchhoff's laws and choosing the variables $x_1 = i_{L1}$, $x_2 = i_{L2}$ and $x_3 = U_C$ gives a three dimensional QP-ODE model:

$$\dot{x}_1 = \frac{1}{L_1}x_3 = x_1\big(\frac{1}{L_1}x_1^{-1}x_3\big) \tag{37}$$

$$\dot{x}_2 = \frac{1}{L_2}x_3 = x_2\big(\frac{1}{L_2}x_2^{-1}x_3\big) \tag{38}$$

$$\dot{x}_3 = \frac{1}{k}x_3^{-\alpha}(-x_1 - x_2) = x_3\big(-\frac{1}{k}x_1x_3^{-\alpha-1} - \frac{1}{k}x_2x_3^{-\alpha-1}\big) \tag{39}$$

The basic algorithm gives the following algebraic dependence:

$$x_3^{\alpha+2} = \Big(\frac{(\alpha+2)(L_1^2 - L_1L_2)}{2kL_2} + \frac{L_1^2}{L_2^2}P\Big)x_1^2 - \Big(\frac{2L_1}{L_2}P + \frac{(\alpha+2)L_1}{k}\Big)x_1x_2 + Px_2^2$$

where $P \in \mathbb{R}$ is an arbitrary parameter. This first integral is equivalent to two non-parametric ones:

$$x_3^{\alpha+2} = \frac{(\alpha+2)(L_1^2 - L_1L_2)}{2kL_2}x_1^2 - \frac{(\alpha+2)L_1}{k}x_1x_2$$

$$x_3^{\alpha+2} = -\frac{\alpha+2}{k}\Big(\frac{L_1}{2}x_1^2 + \frac{L_2}{2}x_2^2\Big)$$

If the capacitor has a linear characteristics (i.e. $\alpha = 0$), the latter first integral describes the conservation of the electrical energy since it can be re-written in the form

$$\frac{1}{2}L_1x_1^2 + \frac{1}{2}L_2x_2^2 + \frac{1}{2}kx_3^2 = const.$$

with k being the capacitance.

The MATLAB implementation of this example uses constant values of L_1, L_2, k and α, and gives the invariant in its parametric form.

5.4 Example 4: A computational example with multiple first integrals

Consider the following QP-ODE system:

$$\dot{x}_1 = x_1(x_2) \tag{40}$$

$$\dot{x}_2 = x_2(x_1^2) \tag{41}$$

$$\dot{x}_3 = x_3(-21x_1^{10}x_2^{-7}x_3^{-1} + 24x_1^6x_2^{-5}x_3^{-1}) \tag{42}$$

$$\begin{aligned}\dot{x}_4 = {} & x_4(10x_1^{10}x_2^{21}x_3^2x_4^{-1} + 20x_1^{12}x_2^{20}x_3^2x_4^{-1} - 42x_1^{20}x_2^{13}x_3x_4^{-1} + \\ & +48x_1^{16}x_2^{15}x_3x_4^{-1})\end{aligned} \tag{43}$$

The application of the multiple retrieval algorithm gives back three first integrals:

$$x_4 = x_1^{10} x_2^{20} x_3^2 \tag{44}$$

$$x_3 = 3x_1^8 x_2^{-7} + 4x_1^6 x_2^{-6} \tag{45}$$

$$x_1^2 = 2x_2 \tag{46}$$

The MATLAB implementation of this example uses the multiple retrieval algorithm successfully. The efficiency of the algorithm in the case of multiple first integrals is clearly visible in this example.

6 Conclusions and further work

A numerically effective polynomial-time algorithm is proposed in this paper for the determination of a class of first integrals in quasi-polynomial systems. The algorithm was implemented and tested in the Matlab numeric computational software environment. The operation and the effectivity of the algorithm is illustrated on several examples: a fed-batch fermentation model, a model of a nonlinear electrical circuit, a Rikitake system and a purely numerical example.

Further work will be focused on two directions. Firstly, on the reimplementation of the algorithm in a symbolic computer-algebra system and secondly, on the constructive application of the algorithm in control oriented nonlinear system analysis and feedback design.

7 Acknowledgements

This research was partially supported by the grant no. OTKA T042710 and F046223. The second author is a grantee of the Bolyai János Research Scholarship of the Hungarian Academy of Sciences.

References

[1] *Matlab User's Guide.* The Math Works, Inc., Natick, MA, 2000.

[2] *Symbolic Math Toolbox User's Guide.* The Math Works, Inc., Natick, MA, 2005.

[3] M.J. Ablowitz, A. Ramani, and H. Segur. A connection between nonlinear evolution equations and ordinary differential equations of P-type I. *J. Math. Phys.*, 21:715–721, 1980.

[4] L. Brenig. Complete factorisation and analytic solutions of generalized Lotka-Volterra equations. *Physics Letters A*, 133:378–382, 1988.

[5] L. Brenig and A. Goriely. Universal canonical forms for the time-continuous dynamical systems. *Phys. Rev. A*, 40:4119–4122, 1989.

[6] L. Brenig and A. Goriely. Painlevé analysis and normal forms. In E. Tournier, editor, *London Math. Soc. Lecture Note Series, Vol. 195*, pages 211–237. Cambridge University Press, Cambridge, 1994.

[7] A.J. Van der Schaft. *L2-Gain and Passivity Techniques in Nonlinear Control.* Springer Verlag, Berlin, 2000.

[8] A. Figueiredo, T.M. Rocha Filho, and L. Brenig. Algebraic structures and invariant manifolds of differential systems. *J. Math. Phys.*, 39:2929–2946, 1998.

[9] A. Figueiredo, T.M. Rocha Filho, and L. Brenig. Necessary conditions for the existence of quasi-polynomial invariants: the quasi-polynomial and Lotka-Volterra systems. *Physica A*, 262:158–180, 1999.

[10] T.M. Rocha Filho, A. Figueiredo, and L. Brenig. [QPSI] A Maple package for the determination of quasi-polynomial symmetries and invariants. *Computer Physics Communications*, 117:263–272, 1999.

[11] J. L. Gouzé. Transformation of polynomial differential systems in the positive orthant. Technical report, INRIA, Sophia-Antipolis, 06561, France, 1990.

[12] B. Hernández-Bermejo and V. Fairén. Nonpolynomial vector fields under the Lotka-Volterra normal form. *Physics Letters A*, 206:31–37, 1995.

[13] B. Hernández-Bermejo, V. Fairén, and L. Brenig. Algebraic recasting of nonlinear systems of ODEs into universal formats. *J. Phys. A, Math. Gen.*, 31:2415–2430, 1998.

[14] Alberto Isidori. *Nonlinear Control Systems*. Springer, Berlin, 1995.

[15] S. Labruine and R. Conte. A geometrical method towards first integrals for dynamical systems. *J. Math. Phys.*, 37:6198–6206, 1996.

[16] P. De Leenheer and D. Aeyels. Stabilization of positive systems with first integrals. *Automatica*, 38:1583–1589, 2002.

[17] A. Magyar, G. Szederkényi, and K.M. Hangos. Quasi-polynomial system representation for the analysis and control of nonlinear systems. In P. Horacek, M. Simandl, and P. Zitek, editors, *Proc. of the 16th IFAC World Congress*, pages 1–6, paper ID: Tu–A22–TO/5. Prague, Czech Republic, 2005.

[18] B. Pongrácz, G. Szederkényi, and K. M. Hangos. The effect of algebraic equations on the stability of process systems modelled by differential algebraic equations. In *Proceedings of the 13th European Symposium on Computer Aided Process Engineering - ESCAPE-13*, pages 857–862, Finland, 2003.

[19] F. Schwartz. Symmetries of differential equations: from Sophus Lie to computer algebra. *SIAM Rev.*, 30:450–481, 1988.

[20] G. Szederkényi and K.M. Hangos. Global stability and quadratic Hamiltonian structure in Lotka-Volterra and quasi-polynomial systems. *Physics Letters A*, 324:437–445, 2004.

[21] G. Szederkényi, K.M. Hangos, and A. Magyar. On the time-reparametrization of quasi-polynomial systems. *Physics Letters A*, 334:288–294, 2005.

[22] G. Szederkényi, M. Kovács, and K. M. Hangos. Reachability of nonlinear fed-batch fermentation processes. *International Journal of Robust and Nonlinear Control*, 12:1109–1124, 2002.