École Polytechnique

BACHELOR THESIS IN COMPUTER SCIENCE

Exploiting Conservation Laws to assess Stuctural Identifiability faster.

Author:

Stefan Vayl, École Polytechnique

Advisor:

Gleb Pogudin, LIX

Academic year 2022/2023

Abstract

Large models described by differential equations are actively used in life science and engineering. The size of a model often makes its analysis and usage computationally expensive. It has been observed that in many models in the literature one can find linear combinations of states which are constant on a trajectory e.g. Conservation of Moment, Mass or Energy. In such cases one can apply technique which is called model reduction or equation elimination, and it can be used to simplify systems of differential equations by removing redundant equations and variables. By identifying sets of linearly dependent equations, one can then proceed with determination of appropriate substitution which further reduces dimension of the model.

We apply such a reduction process in the context of structural parameter identifiability, and show that it can dramatically speed up this, oftentimes quite heavy, computation. Furthermore, in this context, we extend the classical reduction using conservation laws to a conservation laws involving integrals of the observed states. We implemented these algorithms in an open-source package StructuralIdentifiability.jl and we demonstrated their applicability on a series of models from the literature.

Contents

1 Background setting and tools.

The research on this topic was designed and implemented in Julia programming language as a part of StructuralIdentifiability.jl package. Together with Nemo.jl package it became a very powerful tool for researchers and scientists. Let us quickly introduce these two libraries:

1.1 StructuralIdentifiability.jl

StructuralIdentifiability.jl is a package for analyzing the identifiability of mathematical models, particularly those described by ordinary differential equations (ODEs). The package is implemented in Julia and provides efficient algorithms for determining whether a model is structurally identifiable. StructuralIdentifiability.jl also includes functions for performing differential elimination for dynamical systems, which can be used to analyze the identifiability of the model and to determine the minimum set of experiments required to identify the model parameters. StructuralIdentifiability.jl is a valuable tool for researchers and practitioners working in system identification, control theory, and related fields. Its efficient implementation in Julia makes it well-suited for large-scale problems, and its integration with other Julia packages, such as Nemo.jl, makes it a powerful tool for analyzing complex systems.

1.2 Nemo.jl

Nemo.jl is a powerful package for symbolic and numerical computation in algebra and number theory. The package provides a Julia interface to the state-of-the-art C library FLINT (Fast Linear Algebra and Number Theory). Nemo.jl provides a flexible and extensible environment for performing advanced computations in algebra and number theory, including polynomial arithmetic, factorization, and modular arithmetic. Nemo.jl can also be used for more advanced computations, such as computing Galois groups and performing arithmetic on elliptic curves. Nemo.jl is a popular choice for researchers and practitioners in algebra and number theory due to its efficiency, versatility, and ease of use.

2 Introduction

Large dynamical models are widely used to described different scientific phenomena. In real life scenarios one of the main problem with such systems is that given sometimes incomplete, partially observed and noisy data one still needs to design, calibrate and test them. This is mostly due to complexity of measurements as they might be very time consuming (observation of some long going/rare processes where one can not set up multiple experiments at the same time for achieving a greater accuracy) or data collection maybe complicated with lack of tools or their inaccuracy[4]. Thus at the the very first stages of development analysis involving only such called input-output relations play a major role in further developments.
One of such analysis is a determination of structural global identifiability of unknown parameters, that is to say if it is possible to uniquely recover parameters of the model having just limited input data. Different algorithms for such an analysis were studied since 1970 and was observed to be very computationally heavy.
One of the most recently developed and efficient tool for such an analysis is a Julia library StructuralIdentifiability.jl[4]. The algorithm which is used there is based on iterative one by one replacing of state variables by output variables (ones which are known by measurements). For example consider the following case:

$$\begin{cases} \dot{x}_1 = ax_2 \\ \dot{x}_2 = bx_1 + c \\ y_1 = x_1 \end{cases}$$

Where x_1 and x_2 are state variables with corresponding derivatives, y_1 is a variable of output meaning that this quantity is possible to be measured and $a, b, c \in \mathbb{C}$ are the unknown parameters.

(Step 1): Differentiate output y_1 and if applicable substitute values from those in base of state variables:

$$\begin{cases} x_1 = y_1 \dot{x}_2 = bx_1 + c \\ \dot{y}_1 = \dot{x}_1 = ax_2 \end{cases}$$

(Step 2): Repeat the process of differentiation and substitution to attempt elimination of remaining state variables:

$$\begin{cases} x_1 = y_1 \\ x_2 = \frac{\dot{y}_1}{a} \\ \ddot{y}_1 = a\dot{x}_2 = abx_1 + ac = aby_1 + ac \end{cases}$$

(Step 3): Once all state variables were eliminated solve the resulting equation with respect to its parameters:

$$\begin{cases} \ddot{y}_1 = aby_1 + ac \\ \dddot{y}_1 = ab\dot{y}_1 \end{cases}$$

Identifiable = $\mathbb{C}(ab, ac)$

The process of elimination of state variables described above is the heaviest part of the whole algorithm in terms of computation.
Given the fact that in some cases computations may take significant time (hours and in rare cases even days) the possible optimization should be, first of all, considered to the most complicated and heavy process.
One of possible solutions which we will describe in this report is known as variable reduction. Its idea derives from the fact that many real life scenarios described with nonlinear dynamic systems include in some form Conservation Laws such as conservation of Mass, Energy or Moment which make some operations on variables being possible.

3 Formal setting and Examples

We follow a simplified version of the formal setting provided by the "Structural Identifiability" problem. Here we have a system of differential equations Σ of the following form:

$$\begin{cases} \dot{x}(t) = f(x(t), u(t)) \\ y(t) = g(x(t)) \end{cases} \tag{1}$$

Where:

1. $x(t)$ are the states of our models that depend on time
2. $y(t)$ are the observations of our model, that is what we measure in our experiment.
3. $u(t)$ are the external well defined function.
4. $\dot{x}_i = f_i(x(t), u(t))$ is the ith state equation of the model.
5. $\dot{y}_i = f_i(x(t))$ is the ith output equation of the model.
6. $g_i(t), f_i(t)$ are rational functions.

To simplify the notations we will now always write x_n instead of $x_n(t)$ and consider indexes of any variable or function to be positive integers ($i \in [1, n], n \in \mathbb{N}^*$) if other local conditions were not mentioned.

Important to mention that the following concepts and algorithms will be described involving computations only for predetermined real valued coefficients.

Now, we would like to provide few examples of such systems for user to better understand environment in which we will work:

1. Model of oscillatory behavior in enzymatic control processes[6] in four state variables , seven coefficients and output in one variable y_1.

$$\begin{cases} \dot{x}_1 = -b * x_1 + 1/(c + x_4) \\ \dot{x}_2 = \alpha * x_1 - \beta * x_2 \\ \dot{x}_3 = \gamma * x2 - \delta * x3 \\ \dot{x}_4 = \sigma * x4 * (\gamma * x2 - \delta * x3)\frac{1}{x_3} \\ y = x1 \end{cases}$$

2. The model which describes the relationship between HIV and the immune system during the natural course of infection and in the context of different antiviral treatment regimes[8] :

$$\begin{cases} \dot{x} = lm - d * x - \beta * x * v \\ \dot{y} = \beta * x * v - a * y \\ \dot{v} = k * y - u * v \\ \dot{w} = c * x * y * w - c * q * y * w - b * w \\ \dot{z} = c * q * y * w - h * z \\ y_1 = w \\ y_2 = z \end{cases}$$

4 Mathematical concept and theory.

In this approach we aim to find combinations of linearly dependent equations to further make straight substitutions and eliminate some variables out of computation scope.

Definition 1 *Set of equations of the form $\dot{x_i} = f_i(x)$ for some $i > 1$ is called linearly dependent if there exist some corresponding coefficients $(k_i)_{i>1} \in \mathbf{R}$ such that the following equality holds:*

$$k_{x_1}\dot{x_1} + \cdots + k_{x_i}\dot{x_i} = 0$$

One observes that the case of system of first order differential equation of the form $\dot{x_i} = f_i(x)$ for $i > 1$ is equipped with a unique property following from linearity.

Definition 2 *Given set of linearly dependent equations $\dot{x_i} = f_i(x)$ for $i > 1$ and set coefficients $(k_i)_{i>1} \in \mathbf{R}$ such that:*

$$k_{x_1}\dot{x_1} + \cdots + k_{x_i}\dot{x_i} = 0$$

*one takes integral of both parts and obtains the expression for the **first integrals**:*

$$k_{x_1}x_1 + \cdots + k_{x_i}x_i = c$$

where $c \in \mathbf{R}$ is an arbitrary constant.

Definition 3 *A valid substitution is expression derived from first integral of linearly dependent ODE system:*

$$k_{x_n}x_n = c - k_{x_l}x_l - \cdots - k_{x_m}x_m = c$$

where $m, n, k \in \mathbf{N}_{>0}$

Let's consider the following simple example of system of differential equations:

$$\begin{cases} \dot{x_1} = x_1 - x_2 \\ \dot{x_2} = x_2 - x_1 \end{cases}$$

One may notice that their sum is equal to zero:

$$\dot{x_1} + \dot{x_2} = x_1 - x_2 + x_2 - x_1 = 0$$

Proceeding further, one may conclude that

$$(x_1 + x_2)' = 0 \implies x_1 + x_2 = c$$

where c is some constant.

Expressing one variable in terms of other and constant, one obtains a valid expression for substitution:

$$x_1 = c - x_2$$

Now, one may apply this substitution to initial system:

$$\begin{cases} x_1 = c - x_2 \\ \dot{x_2} = x_2 - (c - x_2) = 2x_2 - c \end{cases}$$

Without loss of generality, variable x_1 was eliminated and for further computations one may consider just equation of $\dot{x_2}$.

Definition 4 *Maximum solution is a stopping criteria which corresponds to the iterative phase of algorithm when there are exactly zero linearly dependent equations.*

5 Methods and algorithms

The content in this section is organised with respect to implemented in Julia algorithm step by step.
The general concept of the algorithm might be subdivided into few main sections:

1. Determination of existence of linearly dependent equations (Sections 5.1).
2. Determination and expression of first integrals. (Sections 5.2).
3. Reconstruction of original ODE with applied substitutions (Sections 5.3).

Let us take a detailed look into each of these parts with examples.

5.1 Sets of linearly dependent equations

This problem might be addressed in few different implementations, for example, one may try straightforward iterative approach by summing up all the equations and finding those ones which correspond to the condition for linear dependency. This algorithm may work well on simple examples but for more nested ones it gets computationally heavy as it has exponential complexity with respect to the cardinality of the input model which makes it not an optimal solution. Therefore, one may think of more tricky and general approach.
In the following subsections (5.1.1-5.1.3) we would like to discuss approach involving construction of the matrix derived from coefficients of ODE system.

5.1.1 Matrix construction

Further we will consider notion of **monomial** meaning product of powers of some variables with nonnegative integer exponents, for example, x_1x_2 or $k_1x_1^2x_2^3$.

If one considers any state equation of ODE model to be linear with respect to its monomials it was observed that matrix of coefficients of monomials can be constructed the following way:

1. Take as an input initial ODE system.
2. Create a set which contains all the monomials by iterating through each state equation $\dot{x} = f_i(x)$ and collecting its respected monomials.
   Remark: We consider only unique monomials with respect to coefficients meaning that even if, for example, x_1 and x_1^2 are independent terms in some equations they are treated separately despite their respective variable base is the same.
3. Define a rational valued matrix of the size $m \cdot n$ where m is length of the obtained set of monomials, n is number of state equations, so $i \in n$.
4. Fill the created matrix with rational coefficients corresponding to obtained monomials in each state equation.

Consider the example:

$$\begin{cases} \dot{x}_1 = 2x_1^2 - 3x_2x_3 \\ \dot{x}_2 = -6x_1^2 + 6x_2x_3 - 4x_1x_3^3 \\ \dot{x}_3 = 2x_1x_3^3 + x_1^2 \\ y = x_1 + x_2 \end{cases}$$

The corresponding matrix of coefficients then will look the following way:

$$M_1 = \begin{array}{c} \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1^2 & x_2x_3 & x_1x_3^3 \end{array} \\ \begin{pmatrix} 2 & -3 & 0 \\ -6 & 6 & -4 \\ 1 & 0 & 2 \end{pmatrix} \end{array}$$

On example of real and more complicated model:

$$\begin{cases} \dot{x}_1 = -k_1x_1x_2 + k_2x_4 + k_4x_6 \\ \dot{x}_2 = -k_1x_1x_2 + k_2x_4 + k_3x_4 \\ \dot{x}_3 = -k_6x_3x_5 + k_5x_6 + k_3x_4 \\ \dot{x}_4 = k_1x_1x_2 - k_2x_4 - k_3x_4 \\ \dot{x}_5 = -k_6x_3x_5 + k_4x_6 + k_5x_6 \\ \dot{x}_6 = k_6x_3x_5 - k_4x_6 - k_5x_6 \\ y_1 = x_3 \end{cases}$$

The corresponding matrix will look the following way:

$$M_2 = \begin{array}{c} \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} k_1x_1x_2 & k_2x_4 & k_4x_6 & k_3x_4 & k_5x_6 & k_3x_4 & k_6x_3x_5 \end{array} \\ \begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 & 0 & 1 \end{pmatrix} \end{array}$$

Once obtained we can proceed with processing of the matrix.

5.1.2 Finding linearly dependent equations.

Assuming the matrix described previously successfully found one may proceed with determining nullspace of transpose of such a matrix to obtain matrix which will correspond to linear combinations between equations giving zero as a result Considering the first example from previous section (5.1.1):

$$ker \begin{pmatrix} 2 & -3 & 0 \\ -6 & 6 & -4 \\ 1 & 0 & 2 \end{pmatrix}^T = \begin{array}{c} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{array} \begin{pmatrix} 1 \\ \frac{1}{2} \\ 1 \end{pmatrix}$$

This corresponds to the only solution found at the moment which is interpreted as exactly set of linearly dependent equations and their respective coefficients:

$$1\dot{x}_1 + \tfrac{1}{2}\dot{x}_2 + 1\dot{x}_3 = 2x_1^2 - 3x_2x_3 - \tfrac{1}{2}(-6x_1^2 + 6x_2x_3 - 4x_1x_3^3) + 2x_1x_3^3 + x_1^2 = 0$$

Considering the second case from previous section (5.1.1):

$$ker \begin{pmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 & 0 & 1 \end{pmatrix}^T = \begin{array}{c} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{array} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$

The columns of obtained matrix correspond to three solutions which exist at the current step meaning there are three sets of linearly dependent combinations of state equations:
(solution 1): $\dot{x}_5 + \dot{x}_6 = -k_6x_3x_5 + k_4x_6 + k_5x_6 + k_6x_3x_5 - k_4x_6 - k_5x_6 = 0$
(solution 2): $\dot{x}_2 + \dot{x}_4 = -k_1x_1x_2 + k_2x_4 + k_3x_4 + k_1x_1x_2 - k_2x_4 - k_3x_4 = 0$
(solution 3): $\dot{x}_1 + \dot{x}_3 + \dot{x}_4 - \dot{x}_5 = k_1x_1x_2 + k_2x_4 + k_4x_6 - k_6x_3x_5 + k_5x_6 + k_3x_4 + k_1x_1x_2 - k_2x_4 - k_3x_4 + k_6x_3x_5 - k_4x_6 - k_5x_6 = 0$

Important to consider the special case with possibly occurring fractions as our definition of monomials restricts us to use of only positive exponents of variables without possibility to construct monomials which have some state x_i in denominator. In the following section we will describe algorithm for such a case.

5.1.3 Case with arbitrary fractions

Previously we discussed examples which don't include any fractions except for rational coefficients. If that is the case one may apply previously described algorithm but such an approach has a couple of inconveniences in Julia implementation. Polynomial structures predefined in Nemo.jl[5] constructed the way that once one defines any polynomial which contains fraction it will bring it to the common denominator. Under this condition the good approach will be to find the least common denominator and multiply right hand side of each state equation by the result. So, further computations will be made in basis of this common denominator and will preserve linear combinations we are aiming to find but will not affect any further computations. As a consequence one would obtain modified set of state equations which will be free of fractions and can be used for previously described algorithm of matrix construction.
Consider the example:

$$\begin{cases} \dot{x}_1 = \frac{x_1}{x_2} + \frac{x_2}{x_3} \\ \dot{x}_2 = -2\frac{x_1}{x_2} - \frac{x_2}{x_3} - \frac{x_1}{x_3} \\ \dot{x}_3 = \frac{x_1}{x_2} + \frac{x_1}{x_3} \\ y = x_1 + x_2 \end{cases}$$

Under initial approach, following previously described steps for obtaining and interpreting matrix of coefficients one would obtain the following:

$$M = \begin{array}{c} \\ \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{array} \begin{array}{c} \begin{array}{ccc} \frac{x_1}{x_2} & \frac{x_2}{x_3} & \frac{x_1}{x_3} \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ -2 & -1 & -1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

The obtained least common denominator obtained from collecting all the denominators (x_2, x_3) is equal to x_2x_3.
Multiplying each state equation's rhs by lcm we obtain:

$$\begin{cases} \dot{x}_1 = x_1x_3 + x_2^2 \\ \dot{x}_2 = -2x_1x_3 - x_2^2 - x_1x_2 \\ \dot{x}_3 = x_1x_3 + x_1x_2 \\ y = x_1 + x_2 \end{cases}$$

Corresponding coefficient matrix for this modified ODE system:

$$M_{new} = \begin{array}{c} \\ \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{array} \begin{array}{c} \begin{array}{ccc} x_1x_3 & x_2^2 & x_1x_2 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ -2 & -1 & -1 \\ 1 & 0 & 1 \end{pmatrix} \end{array}$$

Observe that, despite different labeling for columns, matrices M and M_{new} are equal. So, obtained nullspaces of transposes are equal as well and the same equations will have be linearly dependent, that is to say that modifications we made specifically for interpretation in Julia doesn't affect the whole algorithm:

$$ker(M^T) = ker(M_{new}^T) = \begin{array}{c} \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{array} \begin{array}{c} \\ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \end{array}$$

The corresponding solution:

$$\dot{x}_1 + \dot{x}_2 + \dot{x}_3 = x_1x_3 + x_2^2 - 2x_1x_3 - x_2^2 - x_1x_2 + x_1x_3 + x_1x_2 = \\ = \frac{x_1}{x_2} + \frac{x_2}{x_3} - 2\frac{x_1}{x_2} - \frac{x_2}{x_3} - \frac{x_1}{x_3} + \frac{x_1}{x_2} + \frac{x_1}{x_3} = 0$$

5.2 Finding first integral.

Following the definition (2) of the first integrals we can proceed with obtained before results.
Consider previously described example:

$$\begin{cases} \dot{x}_1 = 2x_1^2 - 3x_2x_3 \\ \dot{x}_2 = -6x_1^2 + 6x_2x_3 - 4x_1x_3^3 \\ \dot{x}_3 = 2x_1x_3^3 + x_1^2 \\ y = x_1 + x_2 \end{cases}$$

The obtained solution for finding linearly dependent equations:

$$\dot{x}_1 + \tfrac{1}{2}\dot{x}_2 + \dot{x}_3 = 0$$

Integrating this expression we obtain the following:

$$x_1 + \tfrac{1}{2}x_2 + x_3 = c$$

where $c \in \mathbb{R}$ is some arbitrary constant.
Consider another previously discussed example:

$$\begin{cases} \dot{x_1} = -k_1x_1x_2 + k_2x_4 + k_4x_6 \\ \dot{x_2} = -k_1x_1x_2 + k_2x_4 + k_3x_4 \\ \dot{x_3} = -k_6x_3x_5 + k_5x_6 + k_3x_4 \\ \dot{x_4} = k_1x_1x_2 - k_2x_4 - k_3x_4 \\ \dot{x_5} = -k_6x_3x_5 + k_4x_6 + k_5x_6 \\ \dot{x_6} = k_6x_3x_5 - k_4x_6 - k_5x_6 \\ y_1 = x_3 \end{cases}$$

Here we obtained three different solutions:
(solution 1):

$$\dot{x}_5 + \dot{x}_6 = 0$$
$$x_5 + x_6 = c_1$$

(solution 2):

$$\dot{x}_2 + \dot{x}_4 = 0$$
$$x_2 + x_4 = c_2$$

(solution 3):

$$\dot{x}_1 + \dot{x}_3 + \dot{x}_4 - \dot{x}_5 = 0$$
$$x_1 + x_2 + x_4 + x_5 = c_3$$

where $c_i \in \mathbb{R}, i \in \{1, 2, 3\}$ is arbitrary rational coefficient. Notice that in this example we obtained three different solutions at the moment and some of them intersect with each other. We will consider such a common case in the Section 5.4, right after we will introduce notion and process of the substitution.

5.2.1 Determining variable to be substituted

After obtaining expressions for the first integrals the determination of exact variable to be substituted has to be applied. It was observed that selecting random variable from first integral equation to be expressed and substituted is not the most efficient approach. The algorithm for this operation involves ranking of variables and looks the following way:

1. for each variable $x_{i>0}$ in the first integral find the following parameters:
   (a) Maximal exponent
   (b) Number of occurrences in monomials
   (c) Total occurrences in ODE system
   (d) Maximal rational coefficient

2. Determine the variable with minimal of this parameters with priority of maximal exponent
3. Output this variable

This process allows us to reduce number of operation needed for further substitutions and avoid unnecessary multiplications and exponential computations. Consider the example:

$$\begin{cases} \dot{x}_1 = 2x_1^2 - 3x_2x_3 \\ \dot{x}_2 = -6x_1^2 + 6x_2x_3 - 4x_1x_3^3 \\ \dot{x}_3 = 2x_1x_3^3 + x_1^2 \\ y = x_1 + x_2 \end{cases}$$

The obtained first integral:

$$x_1 + \tfrac{1}{2}x_2 + x_3 = c$$

After search for the parameters corresponding to variables in the first integral $\{x_1, x_2\}$ we obtain the following:
$x_1 : rank_{x_1} = [2, 2, 6, 4]$
$(maximal\_exponent = 2, occurrences\_in\_monomials = 2, total\_occurence = 6, max\_coefficient = 4)$

$x_2 : rank_{x_2} = [1, 2, 3, 6]$
$(maximal\_exponent = 1, occurrences\_in\_monomials = 2, total\_occurence = 3, max\_coefficient = 6)$

$$min(rank_{x_1}, rank_{x_2}) = rank_{x_2}$$

As x_2 has the minimal rank we will further consider it for substitution prior to x_1.

5.2.2 Determining expression for substitutions

Once variable to be substituted was determined, one has to express it from the equation of first integral. Considering the previous example:

$$x_1 + \tfrac{1}{2}x_2 + x_3 = c \implies x_2 = 2(c - x_1 - x_3)$$

5.3 Substitution

After obtaining the previous expression one may proceeds with direct substitution by simply traversing through all the output and state equations and applying the substitution:

$$\begin{cases} \dot{x}_1 = 2x_1^2 - 6(c - x_1 - x_3)x_3 \\ \dot{x}_2 = -6x_1^2 + 12(c - x_1 - x_3)x_3 - 4x_1x_3^3 \\ \dot{x}_3 = 2x_1x_3^3 + x_1^2 \\ y = x_1 + 2(c - x_1 - x_3) \end{cases}$$

The following result is obtained:

$$\begin{cases} \dot{x}_1 = 2x_1^2 - 6cx_3 + 6x_1x_3 - 6x_3^2 \\ \dot{x}_3 = 2x_1x_3^3 + x_1^2 \\ y = -x_1 + 2c - 2x_3 \end{cases}$$

5.4 Case of availability of multiple solutions

In the Section 5.2 we observed the case when there are multiple first integrals available. That is actually a very common example and algorithm should be organized the way it can catch and apply all of them to reach the **maximal solution**. As in that example, we observed that solutions might be not independent meaning that some variables might be involved in different expressions of first integral we implemented recursion concept into algorithm and its final form looks the following way:

(Step 1): Find a coefficient matrix
(Step 2): Find solutions of coefficient matrix
(Step 3): Take the first found solution and extract the expression of single first integral.
(Step 4): Apply the substitution.
(Step 5): Reconstruct the initial ODE system.
(Step 6): Repeat the process with newly obtained system to catch other solution until **maximal solution** was reached.

Consider the example described in the section 5.2:

$$\begin{cases} \dot{x_1} = -k_1x_1x_2 + k_2x_4 + k_4x_6 \\ \dot{x_2} = -k_1x_1x_2 + k_2x_4 + k_3x_4 \\ \dot{x_3} = -k_6x_3x_5 + k_5x_6 + k_3x_4 \\ \dot{x_4} = k_1x_1x_2 - k_2x_4 - k_3x_4 \\ \dot{x_5} = -k_6x_3x_5 + k_4x_6 + k_5x_6 \\ \dot{x_6} = k_6x_3x_5 - k_4x_6 - k_5x_6 \\ y_1 = x_3 \end{cases}$$

First substitution to apply:

$$x_2 = c_1 - x_4$$

Resulting new ode after first iteration:

$$\begin{cases} \dot{x_1} = -k_1x_1(c_1 - x_4) + k_2x_4 + k_4x_6 \\ \dot{x_3} = -k_6x_3x_5 + k_5x_6 + k_3x_4 \\ \dot{x_4} = k_1x_1(c_1 - x_4) - k_2x_4 - k_3x_4 \\ \dot{x_5} = -k_6x_3x_5 + k_4x_6 + k_5x_6 \\ \dot{x_6} = k_6x_3x_5 - k_4x_6 - k_5x_6 \\ y_1 = x_3 \end{cases}$$

First substitution to apply in newly obtained system:

$$x_5 = c_2 - x_6$$

Resulting refreshed ODE system:

$$\begin{cases} \dot{x_1} = -k_1x_1(c_1 - x_4) + k_2x_4 + k_4x_6 \\ \dot{x_3} = -k_6x_3(c_2 - x_6) + k_5x_6 + k_3x_4 \\ \dot{x_4} = k_1x_1(c_1 - x_4) - k_2x_4 - k_3x_4 \\ \dot{x_6} = k_6x_3(c_2 - x_6) - k_4x_6 - k_5x_6 \\ y_1 = x_3 \end{cases}$$

Again another iteration and its substitution:

$$x_1 = c_3 - x_3 - x_4 - x_6$$

Resulting refreshed ODE system:

$$\begin{cases} \dot{x_3} = k_5x_6 + x_6x_3k_6 - x_3k_6c_2 + x_4k_3 \\ \dot{x_6} = -k_5x_6 - x_6x_3k_6 - x_6k_4 + x_3k_6c_2 \\ \dot{x_4} = x_6x_4k_1 - x_6c_1k_1 + x_3x_4k_1 - -x_3c_1k_1 + x_4^2k_1 - x_4k_2 - x_4c_1k_1 - x_4k_1c_3 - x_4k_3 + c_1k_1c_3 \\ y_1 = x_3 \end{cases}$$

6 Extension for the first integrals of y and u variables

Consider the following simple example:

$$\begin{cases} \dot{x}_1 = x_1 + x_2 \\ \dot{x}_2 = x_2 \\ y_1 = -x_1 - x_2 \end{cases}$$

Applying our previous example one will not find any valid solution for such a case. However, observe that:

$$\dot{x}_1 + y = 0$$

In such a case we cannot apply initial algorithm as integral of y is undetermined. It was observed that input equations for determining $y_{i>0}$ integrals might be added with corresponding additional output equations. Before each iteration of initial algorithm :

1. For any existing output equation $y_i = g_i(x)$ add the state equation:

$$\dot{x}_k = g_i(x) \text{ for some new } k \in \mathbb{N}^* \text{ distinct from all } i$$

2. For each new state equation add the new output equation:

$$y_{k>0} = x_k$$

3. Proceed with initial algorithm.

Despite one may notice that in this process number of x variables did not change it was observed that this process still leads to acceleration of computations due to additional output being added. Proceeding with example after described abode tests we will have:

$$\begin{cases} \dot{x}_1 = x_1 + x_2 \\ \dot{x}_2 = x_2 \\ \dot{x}_3 = -x_1 - x_2 \\ y_1 = -x_1 - x_2 \\ y_2 = x_3 \end{cases}$$

The corresponding coefficient matrix:

$$M = \begin{array}{r} \\ \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{array} \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ -1 & -1 \end{pmatrix} \end{array}$$

Its solution:

$$ker(M^T) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The expression for substitution:

$$x_1 = c - x_3$$

With applied substitutions the final result is the following:

$$\begin{cases} \dot{x}_2 = x_2 \\ \dot{x}_3 = -c + x_3 - x_2 \\ y_1 = -c + x_3 - x_2 \\ y_2 = x_3 \end{cases}$$

6.1 Post-processing of obtained ODE system

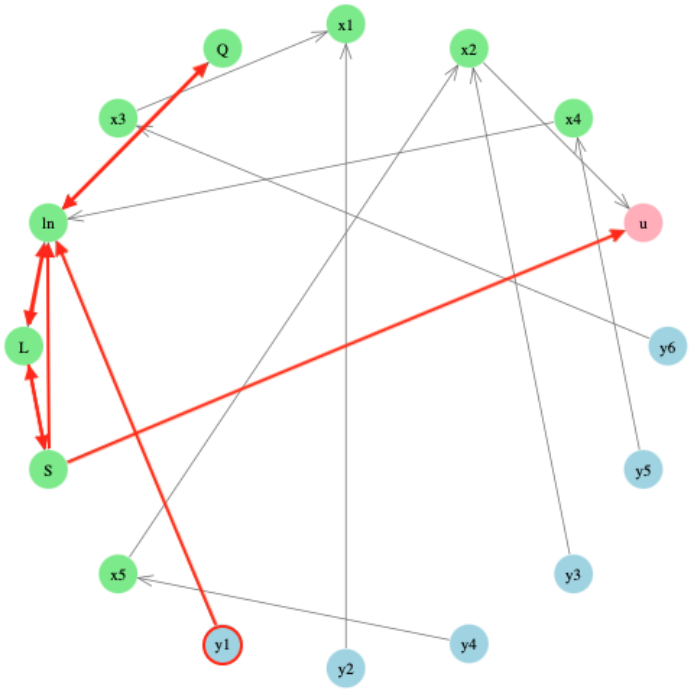
One may observed that by default we always use extensions with y and u integrals and which adds a lot of state and output equations which might eventually not be used in particular cases. Here one may apply cleaning using DFS algorithm:

1. For initial system collect and store all the y's which correspond to output equations.
2. Upon the termination of the algorithm construct the graph for the obtained ODE system with nodes being output and state variables.
3. Apply DFS traversing for each initially stored y
4. Clean the system from equations which didn't pop up while traversing.

In such a way the equations which doesn't have any impact will be cleaned out and will not occur in further computations. Consider as an example the SLIRQ model[3]:

$$\begin{cases} \dot{S} = -bI\frac{S}{S+L+I+R+Q} - u\frac{S}{S+L+I+R+Q} \\ \dot{L} = \frac{bIS}{S+L+I+R+Q} - aL \\ \dot{I} = aL - gI + sQ \\ \dot{R} = \frac{uS}{S+L+I+R+Q} + egI \\ \dot{Q} = (1-e)gI - sQ \\ y_1 = \frac{I}{S+L+I+R+Q} \end{cases}$$

Upon the termination of main algorithm the system as you may see on the following illustration of its graph will contain a lot of unused state and output equations. Starting at y_1 as its our only initial output we consider the red lines to be the path of DFS algorithm where arrows point from variable to variables which have impact on it.



After elimination of unused equations one will obtain the following final output:

$$\begin{cases} \dot{S} = \frac{-b*I*S-uS}{c_1} \\ \dot{L} = \frac{b*I*S-a*L*c_1}{c_1} \\ \dot{I} = sQ - gI + aL \\ \dot{Q} = -sQ - geI + gI \\ y_1 = \frac{I}{c_1} \end{cases}$$

6.2 Validation and Tests

As computations sometimes are quite nested with multiple iterations of algorithm, it is not obvious for observer to understand what exactly happened with initial system and more to this is how to ensure the correctness.
To give user a better view the algorithm stores all the first integrals which were applied and in parallel constructs another technical ODE system which consists of all new added state and output equations but without applied substitutions. Such a system was observed to be useful for the testing.
A good way to ensure the correctness would be computing power series solution for technical ODE system we described before and for the resulting system. In such a case one would obtain solutions for initial and final output equations including substitutions. By comparing the results for initially defined output equations one may conclude that if they are qual the obtained ODE system follows the same rules as initial one and doesn't have any contradictions or mistakes. Consider the example:

$$\begin{cases} \dot{x}_1 = x_2 - x_1 \\ \dot{x}_2 = 2x_1 - x_2 - x_3 \\ \dot{x}_3 = x_3 - x_1 \\ y_1 = x_1 \end{cases}$$

The typical output includes three following parts:

1. The resulting ode

$$\begin{cases} \dot{x}_1 = x_2 - x_1 \\ \dot{x}_2 = -c_1 + 3x_1 \\ y_1 = x_1 \end{cases}$$

2. The dictionary which contains all the substitutions applied during the work of algorithm:

$$\text{c1 => x2 + x3 + x1}$$

3. Technical ODE system on which we don't apply DFS clean up:

$$\begin{cases} \dot{x}_1 = x_2 - x_1 \\ \dot{x}_2 = 2x_1 - x_2 - x_3 \\ \dot{x}_3 = x_3 - x_1 \\ \dot{x}_4 = 1 \\ \dot{x}_6 = x_1 \\ y_1 = x_1 \\ y_2 = x_4 \\ y_3 = x_6 \\ y_4 = x_5 \end{cases}$$

Now if one computes power series solutions for resulting and technical systems the obtained results for y_1 assumed to be the same e.g the obtained system follows the same properties as initial one.
Remark: This process is automatized thankfully to build in tools of StructuralIdentifiability.jl for finding power series solutions.

7 Benchmark and results

The overall complexity of our algorithm is exponential with respect to the size of the input model. Important to mention that indeed approach is iterative and for some stress tests with dimension of hundreds of variables it may take a while to terminate but the real life scenarios very rarely involve that many parameters as it is becoming impossible to construct and analyze such models. Here we want to introduce some results we obtained for real world cases.
Remark: All results correspond to M1 chip with 8Gb RAM and option to use memory swaps.

Model name	T_b	M_b	T_{opt}	N	T_f	M_F
$CRN^{(1)}$	NO	20Gb	<0.5s	3	**<2s**	600Mb
$CRN^{(2)}$	22s	5Gb	<0.5s	3	**21s**	100Mb
$MAPK^{(1)}$	12s	11Gb	<1s	3	**10s**	10Gb
$MAPK^{(2)}$	60s	40Gb+	<1s	3	**20s**	15Gb
$MAPK^{(3)}$	30 min	400Gb+	<1s	3	**27s**	13gb
SIR/SLI	NO	NO	<0.1s	1	**70s**	32Gb
$SLIRQ$	NO	NO	<0.1s	1	**19s**	5Gb

T_b - Initial runtime
M_b - Initial memory usage
T_{opt} - Optimization runtime
N - Number of x states eliminated
T_f - Final runtime including optimization
M_f - Final memory usage
NO - Not observable (at least >1h in terms of time)

7.1 Conclusion

Overall, our algorithm which designed to perform technique called model reduction was successfully implemented as a part of Julia programming language package StructuralIdentifiability.jl[1]. As was shown in the table above it is capable of dramatic increase in performance. Some models, for which time of computation was measured in hours or even days, are able to be computed now in 1-2 minutes . In the same time less complex models can be computed in even shorter time. This gives flexibility to the process of determination of structural identifiability for the unknown parameters in applicable cases as scientists and researchers will have the opportunity to modify, adopt and reconstruct their models much faster by not spending significant time waiting for the computations.

8 Examples

Now let us discuss in details some of these models.

8.1 CRN

This model corresponds to Dynamics of Posttranslational Modification Systems. According to the article[2] Posttranslational modification of proteins (for instance, by phosphorylation or methylation) plays a key role in signal transduction, with disruptions having implications for human health.
As we already used this model as an example (detailed description of algorithm - Section 5.4) we will just introduce here its different combinations which were described in the table above:

$$\begin{cases} \dot{x}_1 = -k_1x_1x_2 + k_2x_4 + k_4x_6 \\ \dot{x}_2 = -k_1x_1x_2 + k_2x_4 + k_3x_4 \\ \dot{x}_3 = -k_6x_3x_5 + k_5x_6 + k_3x_4 \\ \dot{x}_4 = k_1x_1x_2 - k_2x_4 - k_3x_4 \\ \dot{x}_5 = -k_6x_3x_5 + k_4x_6 + k_5x_6 \\ \dot{x}_6 = k_6x_3x_5 + k_5x_6 - k_4x_6 \end{cases}$$

We consider two main sets of outputs:
$CRN^{(1)}$: $y_1 = x_3$ and $y_2 = x_2$
$CRN^{(2)}$: $y_1 = x_3$

8.2 MAPK

Reversible protein phosphorylation on multiple sites is a key regulatory mechanism in most cellular processes[7]. In the following system considered a kinase-phosphatase-substrate system with two sites, under mass-action kinetics, with no restrictions on the order of phosphorylation or dephosphorylation. It has twelve state variables and considers five to six output variables:

$$\begin{cases} \dot{KS}_{00} = -a_{00}KS_{00} + b_{00}KS_{00} + \gamma_{01,00}FS_{01} + \gamma_{10,00}FS_{10} + \gamma_{11,00}FS_{11} \\ \dot{KS}_{01} = -a_{01}KS_{01} + b_{01}KS_{01} + c_{0001}KS_{00} - \alpha_{01}FS_{01} + \beta_{01}FS_{01} + \gamma_{11,01}FS_{11} \\ \dot{KS}_{10} = -a_{10}KS_{10} + b_{10}KS_{10} + c_{0010}KS_{00} - \alpha_{10}FS_{10} + \beta_{10}FS_{10} + \gamma_{11,10}FS_{11} \\ \dot{FS}_{01} = -\alpha_{11}FS_{11} + \beta_{11}FS_{11} + c_{0111}KS_{01} + c_{1011}KS_{10} + c_{0011}KS_{00} \\ \dot{FS}_{10} = a_{00}KS_{00} - (b_{00} + c_{0001} + c_{0010} + c_{0011})KS_{00} \\ \dot{FS}_{11} = a_{01}KS_{01} - (b_{01} + c_{0111})KS_{01} \\ \dot{K} = a_{10}KS_{10} - (b_{10} + c_{1011})KS_{10} \\ \dot{F} = \alpha_{01}FS_{01} - (\beta_{01} + \gamma_{0100})FS_{01} \\ \dot{S}_{00} = \alpha_{10}FS_{10} - (\beta_{10} + \gamma_{1000})FS_{10} \\ \dot{S}_{01} = \alpha_{11}FS_{11} - (\beta_{11} + \gamma_{1101} + \gamma_{1110} + \gamma_{1100})FS_{11} \\ \dot{S}_{10} = -a_{00}KS_{00} + (b_{00} + c_{0001} + c_{00,10} + c_{00,11})KS_{00} - a_{01}KS_{01} + \\ +(b_{01} + c_{0111})KS_{01} - a_{10}KS_{10} + (b_{10} + c_{1011})KS_{10} \\ \dot{S}_{11} = -\alpha_{01}FS_{01} + (\beta_{01} + \gamma_{0100})FS_{01} - \alpha_{10}FS_{10} + (\beta_{10} + \gamma_{1000})FS_{10} - \alpha_{11}FS_{11} + \\ +(\beta_{11} + \gamma_{1101} + \gamma_{1110} + \gamma_{1100})FS_{11} \end{cases}$$

We consider three main sets of outputs:
$MAPK^{(1)}$: $y_1 = K, y_2 = F, y_3 = S_{00}, y_4 = S_{01}, y_5 = S_{10}$ and $y_6 = S_{11}$
$MAPK^{(2)}$: $y_1 = F, y_2 = S_{00}, y_3 = S_{01}, y_4 = S_{10}$ and $y_5 = S_{11}$
$MAPK^{(3)}$: $y_1 = K, y_2 = F, y_3 = S_{00}, y_4 = S_{01} + S_{10}$ and $y_5 = S_{11}$

Let's take the first case $MAPK^{(1)}$ to demonstrate some more details. The corresponding output upon termination of the algorithm looks the following way:

$$\begin{cases}
\dot{S}_{10} = b_{01}KS_{01} + KS_{00}a_{00}K + KS_{00}c_{0001} + KS_{00}c_{0010} + KS_{00}c_{0011} + KS_{00}b_{00} \\
+a_{00}KS_{01} + a_{00}K(KS)_{10} + a_{00}K(KS)_{01} - a_{00}Kc_3 + a_{00}KFS_{01} \\
+a_{00}KF - a_{00}Kc_2 - a_{00}KS_{10} - KS_{01}a_{01} - Ka_{10}S_{10} \\
+c_{1011}K_{S10} + K_{S10}b_{10} + K_{S01}c_{0111} \\
\dot{K} = Ka_{10}S_{10} - c_{1011}K_{S10} - KS_{10}b_{10} \\
\dot{K}_{S01} = b01KS_{01} + \beta_{01}FS_{01} + FS_{11}\gamma_{1101} + KS_{00}c_{0001} - KS_{01}a_{01} \\
-S_{01}\alpha_{01}F \\
\dot{FS}_{01} = FS_{11}\beta_{11} - KS_{00}\alpha_{11}F + KS_{00}c_{0011} - \alpha_{11}KS_{10}F \\
-\alpha_{11}KS_{01}F + \alpha_{11}c_3F - \alpha_{11}FS_{01}F + \alpha_{11}FS_{10} \\
+c_{1011}KS_{10} + KS_{01}c_{0111} \\
\dot{S}_{01} = -FS_{11}\gamma_{1101} - FS_{11}\gamma_{1110} - FS_{11}\beta_{11} - FS_{11}\gamma_{1100} \\
+KS_{00}\alpha_{11}F + \alpha_{11}KS_{10}F + \alpha_{11}KS_{01}F - \alpha_{11}c_3F \\
+\alpha_{11}FS_{01}F - \alpha_{11}FS_{10} \\
\dot{F} = -\beta_{01}FS_{01} + S_{01}\alpha_{01}F - \gamma_{0100}FS_{01} \\
\dot{FS}_{11} = -b_{01}KS_{01} + KS_{01}a_{01} - KS_{01}c_{0111} \\
\dot{KS}_{10} = -FS_{11}\beta_{10} + FS_{11}\gamma_{1110} + KS_{00}c_{0010} + c_1\beta_{10} - K\beta_{10} - Ka_{10}S_{10} - \alpha_{10}FS_{10} - \beta_{10}S_{10} + KS_{10}b_{10} \\
\dot{KS}_{00} = -FS_{11}\gamma_{1000} + FS_{11}\gamma_{1100} + KS_{00}a_{00}K + KS_{00}b_{00} + a_{00}KS_{01} + a_{00}K(KS)_{10} + \\
+a_{00}K(KS)_{01} - a_{00}Kc_3 + a_{00}KFS_{01} + a_{00}KF - a_{00}Kc_2 - a_{00}KS_{10} + \\
+c_1\gamma_{1000} - K\gamma_{1000} + \gamma_{0100}FS_{01} - \gamma_{1000}S_{10}
\end{cases}$$

And output equations will have the following form:

$$\begin{cases}
y_1 = F \\
y_2 = -KS_{00} - S_{01} - KS_{10} - KS_{01} + c_3 - FS_{01} - F + c_2 + S_{10} \\
y_3 = S_{01} \\
y_4 = S_{10} \\
y_5 = KS_{00} + KS_{10} + KS_{01} - c_3 + FS_{01} - S_{10} \\
y_6 = K
\end{cases}$$

As one may notice, during work of algorithm three variables were eliminated out of computational scope with corresponding first integrals:

1. $c_1 = FS_{11} + K + S_{10} + FS_{10}$, eliminate FS_{10}
2. $c_2 = F + S_{00} + S_{01} + S_{11}$, eliminate S_{00}
3. $c_3 = KS_{10} + FS_{01} + KS_{01} - S_{10} - S_{11} + KS_{00}$, eliminate S_{11}

8.3 SLIR/SLI

This is a compartmental model for infectious disease transmission[3] in seven state variables and one output variable:

$$\begin{cases} \dot{Sh} = mh * (Sh + Lh + Ih + Rh) - \frac{bv*Sh*Iv}{(Sh+Lh+Ih+Rh)} - mh * Sh \\ \dot{Lh} = \frac{bv*Sh*Iv}{(Sh+Lh+Ih+Rh)} - ah * Lh - mh * Lh \\ \dot{Ih} = ah * Lh - gh * Ih - mh * Ih \\ \dot{Rh} = gh * Ih - mh * Rh \\ \dot{Sv} = mv * (Sv + Lv + Iv) - \frac{bh*Sv*Ih}{(Sh+Lh+Ih+Rh)} - mv * Sv \\ \dot{Lv} = \frac{bh*Sv*Ih}{(Sh+Lh+Ih+Rh)} - av * Lv - mv * Lv \\ \dot{Iv} = av * Lv - mv * Iv \\ y1 = \frac{Ih}{(Sh+Lh+Ih+Rh)} \end{cases}$$

The resulted output after two iterations:

$$\begin{cases} \dot{Lh} = \frac{(-ah*c_1*Lh-Rh*Iv*bv-Ih*Iv*bv+Iv*bv*c_1-Iv*bv*Lh-mh*c_1*Lh)}{c_1} \\ \dot{Ih} = ah * Lh - gh * Ih - mh * Ih \\ \dot{Rh} = gh * Ih - mh * Rh \\ \dot{Lv} = \frac{(-mv*Lv*c_1-Lv*Ih*bh-Lv*av*c_1-Ih*Iv*bh+Ih*bh*c_2)}{c_1} \\ \dot{Iv} = av * Lv - mv * Iv \\ y1 = \frac{Ih}{c_1} \end{cases}$$

With two corresponding expressions of first integrals::

1. $c1 = Rh + Sh + Ih + Lh$, eliminate Sh
2. $c_2 = Iv + Sv + Lv$, eliminate Sv

9 Future developments

As we mentioned before initially the whole algorithm works only with determined rational coefficients. The reason for this is simply boundaries in the coding part which makes this case special and quite complicated while the whole algorithm remains absolutely the same. So, to go further one may consider the case of construction of coefficient matrix in undetermined coefficients. In some cases it might be useful and output additional solution decreasing computational cost even more dramatically.

10 References

[1] Fork for the repository of structural identifiability with the code. https://github.com/StefanVaylBX2023/StructuralIdentifiability.jl/blob/master/src/first_integrals.jl.

[2] Carsten Conradi and Anne Shiu. Dynamics of posttranslational modification systems: Recent progress and future directions. *Biophysical Journal*, 2018.

[3] Emmanuelle A. Dankwa, Andrew F. Brouwer, and Christl A. Donnelly. Structural identifiability of compartmental models for infectious disease transmission is influenced by data type. *Epidemics*, 41:100643, 2022.

[4] R. Dong, C. Goodbrake, H. Harrington, and G. Pogudin. Differential elimination for dynamical models via projections with applications to structural identifiability. *preprint*, 2021.

[5] Claus Fieker, William Hart, Tommy Hofmann, and Fredrik Johansson. Nemo/hecke: Computer algebra and number theory packages for the julia programming language. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 157–164, New York, NY, USA, 2017. ACM.

[6] Brian C. Goodwin. Oscillatory behavior in enzymatic control processes. *Advances in Enzyme Regulation*, 3:425–437, 1965.

[7] Arjun Kumar Manrai and Jeremy Gunawardena. The geometry of multisite phosphorylation. *Biophysical Journal*, 2008.

[8] Dominik Wodarz and Martin A. Nowak. Specific therapy regimes could lead to long-term immunological control of hiv. *pnas.org*, 1999.