Finding exact linear reductions of dynamical models

Alexander Demin, Elizaveta Demitraki, and Gleb Pogudin

March 30, 2023

HSE University

Some things are smaller than they appear

Exact reduction: a toy example

Setup

Consider a dynamical system in three variables x_1, x_2, x_3:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2 & , \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

Exact reduction: a toy example

Setup

Consider a dynamical system in three variables x_1, x_2, x_3:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2 = (x_2 + 2x_3)^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

Exact reduction: a toy example

Setup

Consider a dynamical system in three variables x_1, x_2, x_3:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2 = (x_2 + 2x_3)^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

Reduction

For $y := x_2 + 2x_3$:

$$\dot{y} = \dot{x}_2 + 2\dot{x}_3 = 4x_3 + 2x_2 = 2(x_2 + 2x_3) = 2y.$$

Exact reduction: a toy example

Setup

Consider a dynamical system in three variables x_1, x_2, x_3:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2 = (x_2 + 2x_3)^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

Reduction

For $y := x_2 + 2x_3$:

$$\dot{y} = \dot{x}_2 + 2\dot{x}_3 = 4x_3 + 2x_2 = 2(x_2 + 2x_3) = 2y.$$

Thus, x_1 and y themselves form a reduced dynamical system:

$$\begin{cases} \dot{y} = 2y, \\ \dot{x}_1 = y^2. \end{cases}$$

Exact reduction: more than one

Reduction to dimension 2 from before:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases} \longrightarrow \begin{cases} \dot{y} = 2y, \\ \dot{x}_1 = y^2. \end{cases}$$

Exact reduction: more than one

Reduction to dimension 2 from before:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases} \longrightarrow \begin{cases} \dot{y} = 2y, \\ \dot{x}_1 = y^2. \end{cases}$$

Can be further refined to a single self-consistent equation:

$$\begin{cases} \dot{y} = 2y, \\ \dot{x}_1 = y^2. \end{cases} \longrightarrow \dot{y} = 2y.$$

Exact reduction: more than one

Reduction to dimension 2 from before:

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases} \longrightarrow \begin{cases} \dot{y} = 2y, \\ \dot{x}_1 = y^2. \end{cases}$$

Can be further refined to a single self-consistent equation:

$$\begin{cases} \dot{y} = 2y, \\ \dot{x}_1 = y^2. \end{cases} \longrightarrow \dot{y} = 2y.$$

In general, an infinite number of linear reductions is possible.

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$
(**f** – polynomial functions)

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$
(**f** – polynomial functions)

Output (one reduction): a linear transformation $\mathbf{y} = \mathbf{x}L$, $L \in \mathbb{C}^{n \times m}$ such that $\dot{y}_1, \ldots, \dot{y}_m$ can be written in terms of $y_1, \ldots, y_m$ (as polynomial expressions)

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$
(**f** – polynomial functions)

Output (one reduction): a linear transformation $\mathbf{y} = \mathbf{x}L$, $L \in \mathbb{C}^{n\times m}$ such that $\dot{y}_1, \ldots, \dot{y}_m$ can be written in terms of $y_1, \ldots, y_m$ (as polynomial expressions)

Example
We had $y = x_2 + 2x_3$, or, equivalently,

$$y = \mathbf{x}L = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$$

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$
($\mathbf{f}$ – polynomial functions)

Output (one reduction): a linear transformation $\mathbf{y} = \mathbf{x}L$, $L \in \mathbb{C}^{n\times m}$
such that $\dot{y}_1, \ldots, \dot{y}_m$ can be written in terms of $y_1, \ldots, y_m$ (as polynomial expressions)

Example
We had $y = x_2 + 2x_3$, or, equivalently,

$$y = \mathbf{x}L = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$$

Example

Any linear first integral is a linear reduction with $\dot{\mathbf{y}} = 0$

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$ ($\mathbf{f}$ – polynomial functions)

Output (one reduction): a linear transformation $\mathbf{y} = \mathbf{x}L$, $L \in \mathbb{C}^{n \times m}$ such that $\dot{y}_1, \ldots, \dot{y}_m$ can be written in terms of $y_1, \ldots, y_m$ (as polynomial expressions)

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$
($\mathbf{f}$ – polynomial functions)

Output (one reduction): a linear transformation $\mathbf{y} = \mathbf{x}L$, $L \in \mathbb{C}^{n \times m}$ such that $\dot{y}_1, \ldots, \dot{y}_m$ can be written in terms of $y_1, \ldots, y_m$ (as polynomial expressions)

Output (many reductions): a sequence of linear transformations

$$\mathbf{y}_1 = \mathbf{x}L_1, \ \ldots, \ \mathbf{y}_\ell = \mathbf{x}L_\ell$$

where each $\mathbf{y}_i = \mathbf{x}L_i$ is a reduction, $0 < m_1 < \ldots < m_\ell < n$, and $L_{i-1} = L_i A_i$ for some A_i.

Exact reduction: formal statement

Input: a system of ODEs $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ in $\mathbf{x} = (x_1, \ldots, x_n)$
($\mathbf{f}$ – polynomial functions)

Output (one reduction): a linear transformation $\mathbf{y} = \mathbf{x}L$, $L \in \mathbb{C}^{n \times m}$ such that $\dot{y}_1, \ldots, \dot{y}_m$ can be written in terms of $y_1, \ldots, y_m$ (as polynomial expressions)

Output (many reductions): a sequence of linear transformations

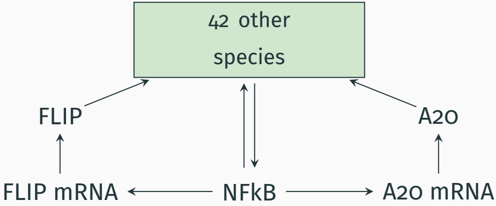
$$\mathbf{y}_1 = \mathbf{x}L_1, \ \ldots, \ \mathbf{y}_\ell = \mathbf{x}L_\ell$$

where each $\mathbf{y}_i = \mathbf{x}L_i$ is a reduction, $0 < m_1 < \ldots < m_\ell < n$, and $L_{i-1} = L_i A_i$ for some A_i.

Such sequence is called a chain of reductions and has finite length

Real-world example

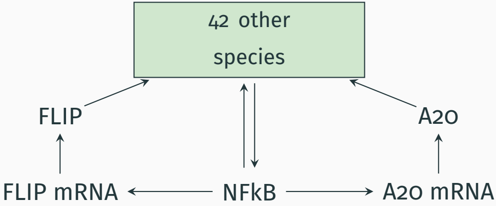
Model of cell death, *Schlieman et al. (2011)*



What is this about?

Real-world example

Model of cell death, *Schlieman et al. (2011)*

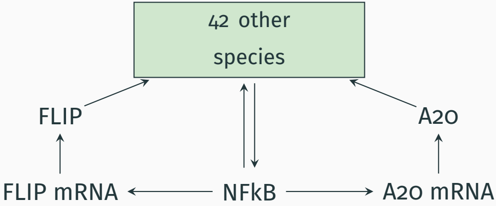


What is this about?

- Two proteins, A20 and FLIP

Real-world example

Model of cell death, *Schlieman et al. (2011)*

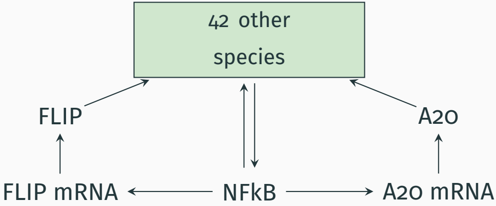


What is this about?

- Two proteins, A20 and FLIP
- The corresponding mRNAs,
  A20 mRNA and FLIP mRNA

Real-world example

Model of cell death, *Schlieman et al. (2011)*

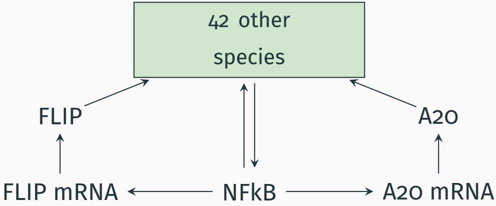


What is this about?

- Two proteins, A20 and FLIP
- The corresponding mRNAs, A20 mRNA and FLIP mRNA
- Nuclear factor NFkB, 47 species in total

Real-world example

Model of cell death, *Schlieman et al. (2011)*



What is this about?

- Two proteins, A20 and FLIP
- The corresponding mRNAs, A20 mRNA and FLIP mRNA
- Nuclear factor NFkB, 47 species in total

A possible reduction

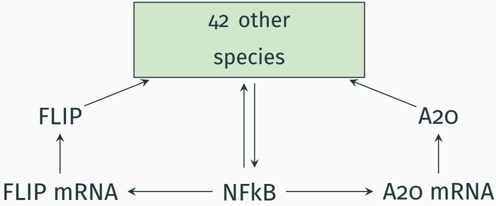
$$\begin{cases} y_1 = \frac{k_6}{k_1}[A20] - \frac{k_5}{k_3}[FLIP], \\ y_2 = k_6[A20\ mRNA] - k_5[FLIP\ mRNA] \end{cases}$$

with the corresponding system

$$\begin{cases} \dot{y}_1 = y_2 + \frac{k_2 k_6}{k_1} - \frac{k_4 k_5}{k_3}, \\ \dot{y}_2 = 0 \end{cases}$$

Real-world example

Model of cell death, *Schlieman et al. (2011)*



What is this about?

- Two proteins, A20 and FLIP
- The corresponding mRNAs, A20 mRNA and FLIP mRNA
- Nuclear factor NFkB, 47 species in total

Plus 16 other reductions (!)

A possible reduction

$$\begin{cases} y_1 = \frac{k_6}{k_1}[A20] - \frac{k_5}{k_3}[FLIP], \\ y_2 = k_6[A20\ mRNA] - k_5[FLIP\ mRNA] \end{cases}$$

with the corresponding system

$$\begin{cases} \dot{y}_1 = y_2 + \frac{k_2 k_6}{k_1} - \frac{k_4 k_5}{k_3}, \\ \dot{y}_2 = 0 \end{cases}$$

Prior results

- *Li and Rabitz (1989, 1991):*
  Approach via Jacobians (on this later!), specific examples

Prior results

- *Li and Rabitz (1989, 1991):*
  Approach via Jacobians (on this later!), specific examples
- *Cardelli, Tribastone, Tschaikowski, Vandin (2017):*
  Fast algorithm with restriction: $y_1, \ldots, y_m$ are sums of disjoint subsets of $x_1, \ldots, x_n$

Prior results

- *Li and Rabitz (1989, 1991):*
  Approach via Jacobians (on this later!), specific examples
- *Cardelli, Tribastone, Tschaikowski, Vandin (2017):*
  Fast algorithm with restriction: $y_1, \ldots, y_m$ are sums of disjoint subsets of $x_1, \ldots, x_n$
  $y_1 = \frac{k_6}{k_1}[A20] - \frac{k_5}{k_3}[FLIP]$

Prior results

- *Li and Rabitz (1989, 1991):*
  Approach via Jacobians (on this later!), specific examples
- *Cardelli, Tribastone, Tschaikowski, Vandin (2017):*
  Fast algorithm with restriction: $y_1, \ldots, y_m$ are sums of disjoint subsets of $x_1, \ldots, x_n$
  $y_1 = \frac{k_6}{k_1}[A20] - \frac{k_5}{k_3}[FLIP] \longrightarrow$ not OK

Prior results

- *Li and Rabitz (1989, 1991):*
  Approach via Jacobians (on this later!), specific examples
- *Cardelli, Tribastone, Tschaikowski, Vandin (2017):*
  Fast algorithm with restriction: $y_1, \ldots, y_m$ are sums of disjoint subsets of $x_1, \ldots, x_n$
  $y_1 = \frac{k_6}{k_1}[A20] - \frac{k_5}{k_3}[FLIP] \longrightarrow$ not OK
- *Perez Verona, Ovchinnikov, Pogudin, Tribastone (2020):*
  Also really fast, but needs a clue: a part of the desired reduction must be given in the input

Prior results

- *Li and Rabitz (1989, 1991):*
  Approach via Jacobians (on this later!), specific examples
- *Cardelli, Tribastone, Tschaikowski, Vandin (2017):*
  Fast algorithm with restriction: $y_1, \ldots, y_m$ are sums of disjoint subsets of $x_1, \ldots, x_n$
  $y_1 = \frac{k_6}{k_1}[A20] - \frac{k_5}{k_3}[FLIP] \longrightarrow$ not OK
- *Perez Verona, Ovchinnikov, Pogudin, Tribastone (2020):*
  Also really fast, but needs a clue: a part of the desired reduction must be given in the input

Focus on finding a single reduction subject to some constraints
$\longrightarrow$ won't autonomously find reductions from our examples

Our results

Our results

We present an algorithm that finds a chain of exact linear reductions without restriction on the coefficients.

The chain will have the maximal possible length.

Real-world example #2

Reaction network

(enzyme deactivation)

$$\begin{array}{ccccc} E + S & \rightleftharpoons & ES & \rightleftharpoons & E + P \\ & & \downarrow & & \\ & & E^* + S & & \end{array}$$

$$E \rightleftharpoons E^*$$

Real-world example #2

Reaction network
(enzyme deactivation)

$$E + S \rightleftharpoons ES \rightleftharpoons E + P$$
$$ES \longrightarrow E^* + S$$
$$E \rightleftharpoons E^*$$

Corresponding ODE system:

$$\begin{cases} [\dot{E}] = 2[ES] + [E^*] - [E][S] - [E][P] - [E], \\ [\dot{S}] = 2[ES] - [E][S], \\ [\dot{P}] = [ES] - [E][P], \\ [\dot{ES}] = [E][S] + [E][P] - 3[ES], \\ [\dot{E^*}] = [E] + [ES] - [E^*] \end{cases}$$

Real-world example #2

Reaction network

(enzyme deactivation)

$$E + S \rightleftarrows ES \rightleftarrows E + P$$

$$ES \downarrow E^* + S$$

$$E \rightleftarrows E^*$$

Corresponding ODE system:

$$\begin{cases} \dot{[E]} = 2[ES] + [E^*] - [E][S] - [E][P] - [E], \\ \dot{[S]} = 2[ES] - [E][S], \\ \dot{[P]} = [ES] - [E][P], \\ \dot{[ES]} = [E][S] + [E][P] - 3[ES], \\ \dot{[E^*]} = [E] + [ES] - [E^*] \end{cases}$$

1. Reduce just a bit

$$\begin{cases} y_1 = E, \\ y_2 = S + P, \\ y_3 = ES, \\ y_4 = E^* \end{cases}$$

Real-world example #2

Reaction network
(enzyme deactivation)

$$E + S \rightleftharpoons ES \rightleftharpoons E + P$$
$$ES \longrightarrow E^* + S$$
$$E \rightleftharpoons E^*$$

Corresponding ODE system:

$$\begin{cases} [\dot{E}] = 2[ES] + [E^*] - [E][S] - [E][P] - [E], \\ [\dot{S}] = 2[ES] - [E][S], \\ [\dot{P}] = [ES] - [E][P], \\ [\dot{ES}] = [E][S] + [E][P] - 3[ES], \\ [\dot{E^*}] = [E] + [ES] - [E^*] \end{cases}$$

1. Reduce just a bit

$$\begin{cases} y_1 = E, \\ y_2 = S + P, \\ y_3 = ES, \\ y_4 = E^* \end{cases}$$

2. Zoom in

$$\begin{cases} y_1 = E + ES, \\ y_2 = S + P + ES, \\ y_3 = E^* \end{cases}$$

Real-world example #2

Reaction network
(enzyme deactivation)

$$E + S \rightleftarrows ES \rightleftarrows E + P$$

$$ES \longrightarrow E^* + S$$

$$E \rightleftarrows E^*$$

Corresponding ODE system:

$$\begin{cases} [\dot{E}] = 2[ES] + [E^*] - [E][S] - [E][P] - [E], \\ [\dot{S}] = 2[ES] - [E][S], \\ [\dot{P}] = [ES] - [E][P], \\ [\dot{ES}] = [E][S] + [E][P] - 3[ES], \\ [\dot{E^*}] = [E] + [ES] - [E^*] \end{cases}$$

1. Reduce just a bit

$$\begin{cases} y_1 = E, \\ y_2 = S + P, \\ y_3 = ES, \\ y_4 = E^* \end{cases}$$

2. Zoom in

$$\begin{cases} y_1 = E + ES, \\ y_2 = S + P + ES, \\ y_3 = E^* \end{cases}$$

3. And zoom in: $y = E + ES - E^*$ ($\dot{y} = -2y$)

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda\in\Lambda} M_\lambda \mathbf{x}^\lambda$

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda \in \Lambda} M_\lambda \mathbf{x}^\lambda$

Example

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda \in \Lambda} M_\lambda \mathbf{x}^\lambda$

Example

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2x_2 + 4x_3 & 8x_3 + 4x_2 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda \in \Lambda} M_\lambda \mathbf{x}^\lambda$

Example

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2x_2 + 4x_3 & 8x_3 + 4x_2 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 & 4 & 8 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_3 + \begin{pmatrix} 0 & 0 & 0 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda \in \Lambda} M_\lambda \mathbf{x}^\lambda$

Example

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2x_2 + 4x_3 & 8x_3 + 4x_2 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 & 4 & 8 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_3 + \begin{pmatrix} 0 & 0 & 0 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda \in \Lambda} M_\lambda \mathbf{x}^\lambda$

Example

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2x_2 + 4x_3 & 8x_3 + 4x_2 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 & 4 & 8 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_3 + \begin{pmatrix} 0 & 0 & 0 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

Algorithm outline: existing tools

Preparation

System $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with polynomial $\mathbf{f}$. Let $J(\mathbf{x})$ be the Jacobian of $\mathbf{f}$.

Li and Rabitz (1991): write $J(\mathbf{x}) = \sum_{\lambda \in \Lambda} M_\lambda \mathbf{x}^\lambda$

Example

$$\begin{cases} \dot{x}_1 = x_2^2 + 4x_2x_3 + 4x_3^2, \\ \dot{x}_2 = 4x_3 - 2x_1, \\ \dot{x}_3 = x_1 + x_2. \end{cases}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2x_2 + 4x_3 & 8x_3 + 4x_2 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

$$J(\mathbf{x}) = \begin{pmatrix} 0 & 2 & 4 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_2 + \begin{pmatrix} 0 & 4 & 8 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} x_3 + \begin{pmatrix} 0 & 0 & 0 \\ -2 & 0 & 4 \\ 1 & 1 & 0 \end{pmatrix}$$

Proposition

For linear forms $y_1, \ldots, y_m$ in $\mathbf{x}$, the following are equivalent:

- $\dot{y}_1, \ldots, \dot{y}_m$ are polynomials in $y_1, \ldots, y_m$;
- the linear span of $y_1, ..., y_m$ is invariant under M_λ for every $1 \leq i \leq m$, $\lambda \in \Lambda$.

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.
Restrict and factor by V

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.
Restrict and factor by V

$$M|_V = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix},$$

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.
Restrict and factor by V

$$M|_V = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}, \; M/V = \begin{pmatrix} 0 & c \\ c & 0 \end{pmatrix}$$

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.
Restrict and factor by V

$$M|_V = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}, \; M/V = \begin{pmatrix} 0 & c \\ c & 0 \end{pmatrix}$$

Algorithm outline: divide and conquer

Long story short. The problem is reduced to

Input: A list of square matrices M_λ, $\lambda \in \Lambda$

Output: A chain of subspaces *invariant* under the matrices

Idea: an invariant subspace if exists, *and then divide-and-conquer*

Example

$$M = \begin{pmatrix} a & b & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & 0 & c \\ 0 & 0 & c & 0 \end{pmatrix}$$

$V = \langle e_1, e_2 \rangle$ is invariant.
Restrict and factor by V

$$M|_V = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}, \; M/V = \begin{pmatrix} 0 & c \\ c & 0 \end{pmatrix}$$

Apply recursively

Algorithm outline: finding one subspace

Many matrices M_λ, $\lambda \in \Lambda$ – need a subspace invariant under all

Algorithm outline: finding one subspace

Many matrices M_λ, $\lambda \in \Lambda$ – need a subspace invariant under all

Algorithm:

1. Find a linear basis of the algebra $\langle M_\lambda \rangle$: multiply matrices by each other until nothing new comes out

Algorithm outline: finding one subspace

Many matrices M_λ, $\lambda \in \Lambda$ – need a subspace invariant under all

Algorithm:

1. Find a linear basis of the algebra $\langle M_\lambda \rangle$: multiply matrices by each other until nothing new comes out
2. Apply the theory of finite-dimensional matrix algebras to find an invariant subspace

Specifics

Algorithm outline: finding one subspace

Many matrices M_λ, $\lambda \in \Lambda$ – need a subspace invariant under all

Algorithm:

1. Find a linear basis of the algebra $\langle M_\lambda \rangle$: multiply matrices by each other until nothing new comes out
2. Apply the theory of finite-dimensional matrix algebras to find an invariant subspace

Specifics

- many matrices, moderate dimension (hundreds)

Algorithm outline: finding one subspace

Many matrices M_λ, $\lambda \in \Lambda$ – need a subspace invariant under all

Algorithm:

1. Find a linear basis of the algebra $\langle M_\lambda \rangle$: multiply matrices by each other until nothing new comes out
2. Apply the theory of finite-dimensional matrix algebras to find an invariant subspace

Specifics

- many matrices, moderate dimension (hundreds)
- but the input is sparse
- and the output is usually *very simple*

The implementation

Package ExactODEReduction.jl, in the Julia language

`https://github.com/x3042/ExactODEReduction.jl`

Running on models from BioModels repository:

Models info		Reductions		Runtime
Dimension	# Models	# Total	# Non-equivalent	Average
2 - 9	44	4.02	1.39	0.6 s
10 - 19	**41**	**8.15**	**2.61**	**0.21 s**
20 - 29	46	9.65	2.13	0.44 s
30 - 39	17	19.41	2.71	1.74 s
40 - 59	25	29.08	6.08	4.58 s
60 - 79	20	37.25	6.95	34.57 s
80 - 99	11	42.91	7.09	96.38 s
100 - 133	4	89.0	21.5	202.52 s

The implementation

Efficiency comes from:

- Sparsity-aware algorithm for finding a basis of an algebra

The implementation

Efficiency comes from:

- Sparsity-aware algorithm for finding a basis of an algebra
- Working over the rationals and postponing passing to the extension as much as possible

The implementation

Efficiency comes from:

- Sparsity-aware algorithm for finding a basis of an algebra
- Working over the rationals and postponing passing to the extension as much as possible
- Modular computation to avoid expression swell

The implementation

Efficiency comes from:

- Sparsity-aware algorithm for finding a basis of an algebra
- Working over the rationals and postponing passing to the extension as much as possible
- Modular computation to avoid expression swell

Features:

- Linear transformations are exact
- Improved interpretability
- Compatibility with the Julia ecosystem

The implementation

Efficiency comes from:

- Sparsity-aware algorithm for finding a basis of an algebra
- Working over the rationals and postponing passing to the extension as much as possible
- Modular computation to avoid expression swell

Features:

- Linear transformations are exact
- Improved interpretability
- Compatibility with the Julia ecosystem

And now software demo

Future work: directions

1. Exact reductions as a preprocessing step, e.g., for checking structural identifiability

Future work: directions

1. Exact reductions as a preprocessing step, e.g., for checking structural identifiability
2. Exact reductions as a way to verify the accuracy of numerical simulations

Future work: directions

1. Exact reductions as a preprocessing step, e.g., for checking structural identifiability
2. Exact reductions as a way to verify the accuracy of numerical simulations
3. Exact reductions for other types of structured dynamical systems, such as, e.g., *graph-based models*

Thank you !

..and my supervisors