École Polytechnique

BACHELOR THESIS IN COMPUTER SCIENCE

Reparametrizing ODE models by scaling

Author:

Milos Oundjian, École Polytechnique

Advisor:

Gleb Pogudin, LIX

Academic year 2023/2024

Abstract

In this thesis, we present a novel implementation of the Hubert and Labahn algorithm for scaling invariants and symmetry reduction in dynamical systems, utilizing the Julia programming language. Our work extends the algorithm's application to non-identifiable models, offering a tool for the simplification and analysis of complex ordinary differential equation (ODE) models prevalent in the natural sciences. By integrating this implementation with the "StructuralIdentifiability.jl" package, we aim to make it easier for scientist to access this algorithm, enabling researchers without extensive computational backgrounds to leverage model simplification and reparameterization techniques. This approach not only facilitates the numerical solution of intricate models but also contributes to a deeper understanding of the underlying principles governing natural phenomena.

Contents

1 Introduction

Differential equations are a fundamental part of many natural sciences. The fields of Biology and Chemistry use differential models for many things. However, computationally, differential equations are not very easy to solve. In 2013, Evelyne Hubert and George Labahn hypothesized (and maybe implemented I'm not sure I couldn't find their code) an algorithm that would allow simplification of differential equations in order to make them easier to solve.

In their paper "Scaling Invariants and Symmetry Reductions of Dynamical Systems" [2], they discuss the theory behind the algorithm and how it can be implemented.

The main overarching goal of this thesis is to implement an algorithm using scaling invariants and symmetry reduction of dynamical systems to improve the efficiency of solving ODEs computationally. What this means is that given an ODE, say for example the predator-prey model given by the following set of equations:

$$\frac{\mathrm{d}n}{\mathrm{d}t} = n(r(1-\frac{n}{K}-k\frac{p}{n+d})),$$
$$\frac{\mathrm{d}p}{\mathrm{d}t} = sp(1-h\frac{p}{n})$$

In order to do this, I implemented in Julia, the necessary functions, such as getting the HNF multipliers in both row and column form. Most of the references for this. These include the functions [2]

Furthermore, I was (hopefully) able to integrate this algorithm that I had implemented into a more overarching Julia library "StructuralIdentifiability.jl"

2 Background

2.1 Hermite Normal Forms and Hermite Multipliers

2.1.1 Hermite Normal Forms (HNFs)

While different sources may have varying definitions of Hermite Normal Forms, (explain the wikipedia def vs sisrds def here), the

According to [2, SISRDS], an $m \times n$ integer matrix $H = [h_{i,j}]$ is said to be in *column* Hermite Normal Form if there exists an integer r and a strictly increasing sequence $i_1, \dots, i_r$ of pivot rows such that:

(i) The first r columns are non-zero;

(ii) $h_{k,j} = 0$ for $k > i_j$;

(iii) $0 \leq h_{i_j,k} < h_{i,jj}$ when $j < k$.

The authors give a similar definition for the *row* Hermite Normal Form being defined as there existing an integer r and a strictly increasing sequence $j_1, \dots, j_r$ such that:

(i) The first r columns are non-zero;

(ii) $h_{i,k} = 0$ for $k > j_i$;

(iii) $0 \leq h_{k,j_i} < h_{i,j_i}$ when $i < k$.

Note that in this definition has index (i) unchanged for both row and column hermite normal form, and this definition does not seem standard across all sources came across (example here is wikipedia).

The reason that we care so much about the differences between is that our algorithm uses both forms in its different stages and we need to be very careful about which one we choose.

2.1.2 Hermite Multipliers

Any integer matrix A can be transformed via integer row operations (respectively column) to obtain a unique row (respectively column) Hermite Normal Form matrix H. We can encode these row operations (respectively column) into a unimodular integer matrix V such that we have $V \cdot A = H$ (respectively $A \cdot V = H$). This matrix V is sometimes called the Hermite transform. In this paper we will be following [2] and calling in the *Hermite Multiplier* of A.

TODO: Maybe add an example here

2.1.3 Algorithmically determining HNF and Hermite Multipliers

There exists algorithms that can computer Hermite Normal Forms which can be found for example in [1]. The steps and complexity of this algorithm is discussed there as well. However, for the scope of this paper we will not discuss these algorithms in detail.

Effectively, we will use these algorithms as a part of our algorithm. To do this we give this algorithm a specification. The function we care about is in the Julia library `Nemo.jl` and is called `hnf_with_transform`. The specification of this function is as follows:

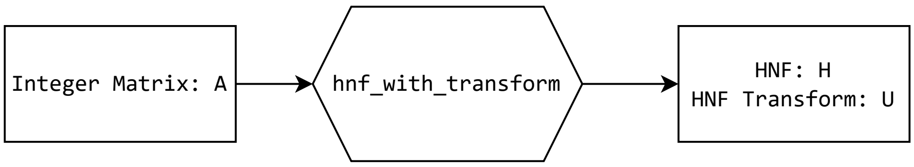

Figure 1: Existing Algorithm Specifications

TODO: Improve this figure to show that it is using row form

This means that in our algorithm, we already have a tool that allows us to calculate row Hermite Normal Form. What we need to do is to find a way to use this tool, for example by doing matrix operations on the input matrix A and on the result from using the function `hnf_with_transform` to be able to create a new function that taking input matrix A, returns the column hermite normal form of A. In the next section we will show how we are able to do this and prove our new method.

The algorithm we implemented is as follows. It takes as input an $m \times n$ integer matrix A and returns H, V where H is the Hermite Normal Form of A and V is the Hermite multiplier.

1. Reverse the rows of A.
2. Transpose A.
3. Get the Hermite Normal Form of the new A and its transpose and its

2.1.4 Practical Implementation

There exists open source libraries that implement the algorithms to calculate the HNF form of a given matrix. For the purpose of this algorithm I have chosen to use the impl

TODO: Proof

2.2 Scalings

A scaling is an operation on the differential equation that changes the

3 Implementation

In this section I will explain with an example the full methodology of the algorithm. The example I use is the the Schackenberg Model for a simple chemical reaction with limit cycle as shown in example 7.4 of [2]. The equation for this model is as follows.

$$\frac{\mathrm{d}x}{\mathrm{d}t} = a - kx + hx^2y$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = b - hx^2y$$

I explain the process to work on this from now on.

Step 1: Form the function F with the Laurent polynomials of the differential equations.

3.1 Example Implementation 1

3.2 Example Implementation 2

4 References

[1] H. Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer Berlin Heidelberg, 2013.

[2] Evelyne Hubert and George Labahn. Scaling invariants and symmetry reduction of dynamical systems. *Foundations of Computational Mathematics*, 13(4):479–516, Aug 2013.

A Appendix