UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MONOMIAL AND OPTIMAL QUADRATIZATION FOR PDES

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

ALBANI CRISTINA OLIVIERI REYES

PROFESOR GUÍA:
GONZALO NAVARRO
PROFESOR GUÍA 2:
GLEB POGUDIN

MIEMBROS DE LA COMISIÓN:
NOMBRE COMPLETO UNO
NOMBRE COMPLETO DOS
NOMBRE COMPLETO TRES

SANTIAGO DE CHILE
2024

Resumen

Las ecuaciones diferenciales parciales (EDP) son esenciales para representar fenómenos y comportamientos dinámicos en áreas científicas, como la física y la ingeniería. Asimismo, los sistemas dinámicos descritos por estas estructuras matemáticas exhiben dinámicas complejas, especialmente cuando las estructuras de las ecuaciones son no lineales. Además, se ha demostrado ampliamente que los sistemas cuadráticos suelen ser más fáciles de analizar, simular, controlar y aprender.

En la misma línea de ideas, esta tesis propone un algoritmo para obtener cuadratizaciones de EDPs. Esta es una técnica en la que un sistema de EDPs con expresiones del lado derecho de grado superior a dos se eleva o transforma a un sistema de EDP con expresiones del lado derecho de grados como máximo cuadráticos. Para lograrlo, se introducen nuevas variables, así como ecuaciones diferenciales parciales correspondientes a estas nuevas relaciones provenientes de la transformación o renombramiento de expresiones. Este procedimiento se ha utilizado en diversos campos científicos, como en el estudio de sistemas dinámicos, en reducción de orden de modelos no lineales, en redes de reacciones químicas, entre muchos otros.

Nuestro algoritmo encuentra una cuadratización para un sistema de EDP garantizando su optimalidad (es decir, garantiza el menor número de variables) dentro del espacio de búsqueda. Para este algoritmo se implementaron dos versiones para la búsqueda de nuevas variables, una con un enfoque *branch-and-bound* y otra con un enfoque *nearest-neighbor*. Finalmente, mostramos resultados del rendimiento de ambas implementaciones utilizando modelos de EDP provenientes de aplicaciones prácticas. A través del enlace `https://github.com/albaniolivieri/pde-quad.git` es posible acceder al paquete de software correspondiente.

Abstract

Partial differential equations are essential to represent phenomena and dynamical behaviors in mathematically oriented scientific fields, such as physics and engineering. Moreover, dynamical systems described by these mathematical structures exhibit complex dynamics, especially when the expression structures are nonlinear. Further, it has been widely demonstrated that quadratic systems are often easier to analyze, simulate, control, and learn.

Therefore, this thesis introduces an algorithm for obtaining quadratizations for Partial Differential Equations (PDE). This is a technique where a PDE system with right-hand side expressions of degrees greater than two is lifted or transformed into a PDE system with right-hand side expressions of at most quadratic degrees. To accomplish this, new variables and partial differential equations corresponding to the relations of these auxiliary variables are introduced. This procedure is used in several scientific fields, such as dynamical systems, nonlinear model order reduction, and chemical reaction networks, among many others.

Our algorithm finds a quadratization for a given PDE system and guarantees its optimality (i.e., it introduces the least number of variables) within the search space. Two versions of this algorithm were implemented to search for new variables, one with a branch-and-bound approach and the other with a nearest-neighbor approach. We show some results of the performance of both implementations and benchmarks using PDE models obtained from practical applications. Our software implementation is available in `https://github.com/albaniolivieri/pde-quad.git`.

A mis padres y mis hermanas, esto es tanto mío como de ustedes.

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Introduction

One of the most important challenges in numerical analysis is the study of complex dynamical processes described by Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs). In particular, nonpolynomial and nonquadratic PDEs describe complex dynamical processes in science and engineering. Some examples are the cubic FitzHugh-Nagumo model [5], which explains the activation and deactivation dynamics of a spiking neuron; the also cubic Brusselator model [33], used to predict oscillations in chemical reactions; and the quartic model of the nonadiabatic tubular reactor [31], which describes the evolution of the species concentration and temperature.

In addition, using a specific model structure, such as quadratic PDE systems, is of broad interest in many scientific fields. One of them is the area of Model Order Reduction (MOR), where transforming or lifting such systems into quadratic form (as introduced by Gu [20]) has been used to obtain better variables for several purposes: for learning low-order polynomial reduced models of complex nonlinear systems [39], for developing system-theoretic MOR [4], and for conducting studies of data-driven MOR [30], among others.

Also, in the context of MOR, the fluid dynamics community has shown increasing interest in transforming a given system into a quadratic form. Balajewicz et al. [3] benefit from a quadratic structure to develop a projection-based reduced order model (ROM) for the Navier-Stokes equations [42], PDEs that describe the flow of incompressible fluids. Furthermore, Guillot et al. [22] use this quadratic transformation for developing a Taylor series–based continuation algorithm for finding the solution set of a given algebraic system.

Another field that benefits from this structure is analog computation with chemical reaction networks. Lifting differential equation models into a quadratic form allows transforming a given chemical reaction network into a bimolecular one, thus obtaining an elementary chemical reaction representation. This result was fundamental in proving the Turing completeness of elementary chemical reactions ([24, 17, 6]).

Several works mentioned before have encountered efficiency problems when finding these

new transformation variables, especially when handling PDE systems. In contrast with the case of ODEs, where finding a quadratic form of these systems has been widely studied, methods for obtaining this structure for PDEs have remained unexplored.

In this thesis, we discuss the concept and study of quadratization of PDEs. This refers to the process of finding a transformation that turns a PDE system with nonpolynomial or higher-degree polynomial right-hand side equations into a quadratic system. To obtain a quadratic form, one may need to add new auxiliary variables to the original PDE, where the set of variables introduced is called a quadratization. The main problem considered in this thesis is the optimal monomial quadratization of PDEs.

To gain some insight into this, let us first overview the quadratization problem for ODE systems [10]. As with PDEs, the ultimate goal of the quadratization procedure for ODEs is to decrease the right-hand side degree to at most two. To achieve this result, we introduce to rewrite the original ODE system, resulting in a new system with lower right-hand side degrees. For example, if we have the following ODE:

$$x' = x^5, \tag{1.1}$$

one of the possible quadratizations is obtained by defining the new variable $y := x^4$. With the definition of y, we also need to add a new differential equation corresponding to this new relation. Thus, we obtain:

$$x' = xy \quad \textit{and} \quad y' = 4x^3x' = 4x^8 = 4y^2. \tag{1.2}$$

In the previous example, every solution of the original system is the x-component of some solution of Eq. (1.2). Additionally, we call a quadratization optimal if it introduces the minimum number of variables, and a monomial quadratization if the variables introduced are one-term polynomials. Our example shows that our quadratization is optimal as it introduces only one new variable.

Now, the method for finding an optimal monomial quadratization for a PDE should bear some differences from the one presented above given the mathematical nature of PDEs. A PDE is defined by independent variables (x, y, ...) and some unknown functions of these variables ($u_1(x, y, ...), \dots u_n(x, y, ...)$). A PDE is an equation that imposes relations between the independent variables, the functions $u_1, \dots u_n$, and their partial derivatives [40].

Due to the multivariate nature of PDEs, this new problem is more complex compared to the ODE case, since the partial derivatives of both the unknown functions and the new variables are part of the quadratic transformation. To the best of our knowledge, no algorithm for finding an optimal monomial quadratization for PDE systems exists to date. This is the main goal of this work.

Let us illustrate the PDE case of the quadratization problem with an example. Let our PDE be defined by the unknown function $u(t, x)$ (dependent on time and space) and the following equation:

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial x} u^2. \tag{1.3}$$

To quadratize Eq. (1.3) we can introduce the variable $y := u^2$. Since our differential equation is a PDE, we can calculate y's first derivative with respect to x:

$$\frac{\partial y}{\partial x} = 2\frac{\partial u}{\partial x} u. \tag{1.4}$$

As in the ODE case, we add a new differential equation corresponding to y. Now we write:

$$\frac{\partial y}{\partial t} = 2\frac{\partial u}{\partial x} u^3 = 2\frac{\partial u}{\partial x} uy = \frac{\partial y}{\partial x} y \quad \textit{and} \quad \frac{\partial u}{\partial t} = \frac{\partial u}{\partial x} y. \tag{1.5}$$

Looking at the transformed system in Eq. (1.5), we notice that the rewritten form of the equation Eq. (1.3) and the new differential equation are both quadratic. Then, we say that the set $\{u^2\}$ is an optimal quadratization for the example PDE in Eq. (1.3). On the other hand, we could have accomplished a quadratic transformation for the same system using different variables. For example, the set $\left\{\frac{\partial u}{\partial x} \cdot u, u^3\right\}$ is also a quadratization, but with a higher number of variables, i.e., a nonoptimal quadratization.

This thesis centers on designing and constructing an algorithm for finding optimal quadratizations for PDE systems given a number of differentiations for the auxiliary variables. In particular, the implementation of this algorithm outputs a quadratization with an optimal number of introduced variables, with the search space defined by the decomposition of the polynomial terms from the nonquadratic PDE right-hand sides. Lastly, we show some benchmarks related to performance using practical PDE examples found in the literature.

1.2 Problem Statement

In the following, we give a preliminary definition of quadratization for PDEs in order to state more clearly the problem we want to solve. A formal definition will be presented in Section 2.2.2.

Definition 1.1 *(Preliminary definition of quadratization for PDEs)*

For simplicity, let us consider a PDE with a bivariate unknown function

$$\frac{\partial u}{\partial t} = F\left(u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, \ldots, \frac{\partial^h u}{\partial x^h}\right), \tag{1.6}$$

where u is the unknown function depending on variables t and x, and h is the order of the differential equation (highest order of partial derivatives). We call the set comprised of u and its partial derivatives $\boldsymbol{U}$ and write $F(\boldsymbol{U})$. Then, let us consider a set $\boldsymbol{Y}$ of n variables consisting of the following elements:

$$y_1 = g_1\left(\boldsymbol{U}\right), \ldots, y_n = g_n\left(\boldsymbol{U}\right), \tag{1.7}$$

where $g_1, \ldots, g_n$ are monomials. We also consider the partial derivatives with respect to the second independent variable up to an order k:

$$\frac{\partial y_1}{\partial x} = \frac{\partial g_1}{\partial x}, \ldots, \frac{\partial^k y_1}{\partial x^k} = \frac{\partial^k g_1}{\partial x^k}, \ldots, \frac{\partial y_n}{\partial x} = \frac{\partial g_n}{\partial x}, \ldots, \frac{\partial^k y_n}{\partial x^k} = \frac{\partial^k g_n}{\partial x^k}. \tag{1.8}$$

We call this set $\boldsymbol{D}^k_{Y_x}$. Then, the set $\boldsymbol{Y}$ of new variables in Eq. (1.7) are considered a quadratization of Eq. (1.6) if:

- *There exists an integer k and polynomial p of maximum degree less or equal than two such that:*
$$\frac{\partial u}{\partial t} = p\left(\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{D}^k_{Y_x}\right). \tag{1.9}$$
- *As we need to add differential equations for each new variable introduced to state their relations (explained by Gu [20]), polynomials $q_1 \ldots q_n$ must also exist with degree at most two such that:*
$$\frac{\partial y_i}{\partial t} = q_i\left(\boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{D}^k_{Y_x}\right) \text{ for every } 1 \leq i \leq n, \tag{1.10}$$

where n is the order of the quadratization.

If we call m the minimum number of new variables needed to obtain a quadratization for Eq. (1.6), then we say that a quadratization of order m is optimal. Knowing this definition, it is natural to ask ourselves the following questions: How to verify if a set of variables is a quadratization for Eq. (1.6)? Moreover, how to uncover this particular set? And lastly, how to guarantee that it is optimal?

To gain insight into the questions above, we will present some PDE examples and attempt to find quadratizations for them. Consider the PDE with time (t) and space (x) variables:

$$\frac{\partial u}{\partial t} = \left(\frac{\partial u}{\partial x}\right)^3. \tag{1.11}$$

We first need to propose quadratization candidates, so we decompose the monomial in Eq. (1.11) right-hand side to obtain the decompositions in which the expression turns quadratic if we rename coupled variables. Among these decompositions, we choose the new variable in Eq. (1.12) and calculate its first derivative in x:

$$w = \left(\frac{\partial u}{\partial x}\right)^2, \tag{1.12}$$

$$\frac{\partial w}{\partial x} = 2\frac{\partial u}{\partial x}\frac{\partial^2 u}{\partial x^2}. \tag{1.13}$$

We must check if this new variable is a quadratization for Eq. (1.11). To do this, we verify if we obtain only quadratic right-hand sides after performing substitutions in both the original PDE and the new differential equation we introduce. We follow this step in our example:

$$\frac{\partial w}{\partial t} = 2\frac{\partial u}{\partial x}\left(3\left(\frac{\partial u}{\partial x}\right)^2\frac{\partial^2 u}{\partial x^2}\right) = 3\left(\frac{\partial u}{\partial x}\right)^2\left(2\left(\frac{\partial u}{\partial x}\right)^2\frac{\partial^2 u}{\partial x^2}\right) = 3\frac{\partial w}{\partial x}w, \tag{1.14}$$

$$\frac{\partial u}{\partial t} = w\frac{\partial u}{\partial x}. \tag{1.15}$$

Now, we notice that all the system equations are quadratic on their right-hand side with only one new auxiliary variable. Then we say that $\left\{\left(\frac{\partial u}{\partial x}\right)^2\right\}$ is an optimal quadratization. If the quadratization set we picked consisted of two or more variables, guaranteeing optimality would not have been trivial, as we would need to demonstrate that no quadratization with fewer variables exists.

By analyzing the resulting quadratization in the example above, we notice that as in the first PDE shown in Eq. (1.3), we had to introduce only one auxiliary variable and its first derivative in x. On the other hand, suppose we keep trying to find a quadratization with other new variables for this PDE. In that case, we obtain that at least three more sets are quadratizations: $\left\{\left(\frac{\partial u}{\partial x}\right)^3, \left(\frac{\partial u}{\partial x}\right)^4\right\}$, $\left\{\left(\frac{\partial u}{\partial x}\right)^3, \frac{\partial^2 u}{\partial x^2}\left(\frac{\partial u}{\partial x}\right)^3\right\}$, $\left\{\left(\frac{\partial u}{\partial x}\right)^3, \frac{\partial^2 u}{\partial x^2}\frac{\partial u}{\partial x}\right\}$. Then, when considering an algorithmic solution for finding an optimal quadratization, it is clear that we must thoroughly think through its design, as we would like to avoid an exhaustive search through the potentially numerous quadratization candidates.

Let us try to find a quadratization for a more complex example, a PDE model that describes the solar wind speed [28]. This PDE depends on r, the radial distance from the Sun, and ϕ, the Carrington longitude in the heliographic (HG) coordinate system. The governing equation is:

$$\frac{\partial v}{\partial r} = \omega\frac{\partial v}{\partial \phi}\frac{1}{v}, \tag{1.16}$$

with ω a constant. First, we notice that the right-hand side of Eq. (1.16) is not polynomial. For this case, we can attempt the same quadratization trick of adding a new variable and its differential equation to obtain a polynomial system (this procedure is commonly known as polynomialization). If we introduce the variable $y = \frac{1}{v}$, we get the system:

$$\frac{\partial v}{\partial r} = \omega\frac{\partial v}{\partial \phi}y, \tag{1.17}$$

$$\frac{\partial y}{\partial r} = -\omega\frac{\partial v}{\partial \phi}y^3. \tag{1.18}$$

which is a polynomial system of degree greater than two. Then, we can attempt to find a quadratization by introducing the second new auxiliary variable $w = y^2$ with its first derivative in ϕ:

$$\frac{\partial w}{\partial \phi} = 2y\frac{\partial y}{\partial \phi} = -2y^3\frac{\partial v}{\partial \phi}.$$

The final PDE system with this new variable is:

$$\frac{\partial v}{\partial r} = \omega\frac{\partial v}{\partial \phi}y, \tag{1.19}$$

$$\frac{\partial y}{\partial r} = -\omega\frac{\partial v}{\partial \phi}y^3 = \frac{\omega}{2}\frac{\partial w}{\partial \phi} \tag{1.20}$$

$$\frac{\partial w}{\partial r} = -2\omega y^4\frac{\partial v}{\partial \phi} = \omega y\frac{\partial w}{\partial \phi}. \tag{1.21}$$

From the resulting equations, we see that $\{y^2\}$ is a quadratization for the polynomial system shown in Eq. (1.17) and Eq. (1.18). Now, there is a quadratization found for this PDE in the literature using a dimension agnostic quadratization procedure [9]. By comparing the results from this work with our findings we can outline some differences; first, they had to discretize the original equation to attempt a quadratization, and second, the quadratization they found was of size four, which provides some clues on the advantages of the method we are proposing.

Another insight we gain from this example is that quadratization for nonpolynomial PDEs is also an area of interest, particularly for rational functions. Therefore, finding a quadratization algorithmically for this type of PDE becomes part of the scope of the problem we want to tackle.

Considering the lower order of the resulting quadratizations, the simplicity of the procedure, and how it compares with results in the literature, we propose the methodology used in the examples above to find a quadratization for a PDE system, adding the implementation of a search optimization algorithm. Moreover, we suspect this approach is more likely to yield an optimal monomial quadratization for a given PDE. Given their relevance in practical models, we also aim to find a solution that can handle rational functions. Then, we state the main problem we want to address in an input/output form.

Input: A PDE system of the form Eq. (1.6) and an integer p representing the number of partial differentiations to calculate for the new variables.

Output: A quadratized form of the PDE system with new variables and their partial derivatives within the right-hand side equations.

1.3 Research Hypothesis

- **First hypothesis**: In relevant problems, it is possible to develop an algorithm that discovers quadratizations for PDE systems that surpass the order and time performance of the state-of-the-art approaches.
- **Second hypothesis**: It is possible to design and develop a practically efficient algorithm that finds an optimal (lowest order) quadratization for a PDE system given a defined order of partial derivatives for the new variables.

1.4 Research Objectives

1.4.1 Main Goal

Develop and test an algorithm that finds an optimal monomial quadratization for a given PDE system and a defined number of partial differentiations for the new variables.

1.4.2 Specific Goals

- Design an algorithm for verifying if a given set of monomial variables and its partial derivatives up to an order p is a quadratization for a PDE system.
- Design searching techniques for finding an optimal quadratization (set of new variables) among all the possible ones within the search space given by the monomial decompositions of a PDE system.
- Implement and integrate the algorithms described above into software.
- Test its performance in terms of time efficiency and quality of results with practical PDE examples. Also, compare outputs using both the proposed algorithm and the available methods for quadratization for PDEs found in the literature.
- Enhance the existing algorithm by incorporating additional functionalities to handle a wider range of non-linear functions, such as rational functions.

1.5 Methodology

This section outlines the methodology employed to achieve the specific objectives of the research. In summary, the study comprises the following steps:

1. Reach a full understanding through an intensive literature review of the quadratization algorithm for the ODE case to use it as a starting point for the design of the new algorithm.

2. Design an algorithm for finding optimal monomial PDE quadratizations. To divide the problem, the development of the algorithm is divided into three different parts:
    2.1 Design, implement and test a module that verifies if a set of variables is a quadratization.
    2.2 Develop the module that proposes variable candidates for a quadratization.
    2.3 Develop an algorithm that follows a search design paradigm to find the optimal quadratization within all the possible ones. For the algorithm search technique, the approaches to use are branch-and-bound and nearest-neighbor algorithms.
3. Compare the performance of the implemented algorithm with the results from the state-of-the-art approaches for finding quadratizations for PDEs.
4. Test and evaluate results obtained from practical examples of PDE models.
5. Develop improvements, functionalities, and extensions to the algorithm.

1.6 Outline

Chapter 2 discusses the theoretical background and related work that underlies our proposed methodology. Chapter 3 explores the algorithmic foundations and implementation of our software.

In Chapter 4, we discuss the experiments and present the main results from our benchmarks for the quadratization algorithms using the software implementation available in `https://github.com/albaniolivieri/pde-quad.git`. We analyze the performance of our software across a range of PDE examples and give insights gained from our experiments.

Finally, in Chapter 6, we summarize our research findings, discuss the limitations of our approach, suggest areas for improvement, and outline directions for future research.

Chapter 2

Background and Related Work

This chapter provides an overview of the scientific disciplines related to this work. We review the core concepts of symbolic computation and partial differential equations, which are central to this thesis. Next, we further explore the fundamentals of this investigation by describing the definitions of quadratization, Gaussian elimination, depth-first and best-first searches, the branch-and-bound framework, the incremental nearest-neighbor algorithm, Gröbner bases, and Buchberger's algorithm. We also clarify the notation used throughout the presented document. Finally, we review the related work and state-of-the-art quadratization methods.

2.1 Scientific Disciplines

2.1.1 Symbolic Computation

Symbolic computation is a scientific area related to the fields of mathematics and computer science that is concerned with the study, development, implementation, and application of algorithms that manipulate mathematical expressions and other mathematical objects.

In contrast to numerical methods, symbolic methods treat objects that are either formal expressions or are algebraic in nature [37]. While numerical computing helps find solutions to desired precision for problems that do not have exact solutions, in many cases, numerical methods will not give sufficient information about the nature of the problem. On the other hand, symbolic methods can offer more insight into the problem we are trying to solve. Another advantage is that, while numerical methods may simply fail to compute correct results, symbolic methods yield closed or explicit formulas.

2.1.2 Partial Differential Equations

A partial differential equation (PDE) [16] is an equation involving an unknown function of two or more variables and some of its partial derivatives. Partial differential equations describe physical systems, such as solid and fluid mechanics, the evolution of populations and disease, and mathematical physics. They are foundational in the modern scientific understanding of sound, heat, diffusion, electrostatics, electrodynamics, thermodynamics, fluid dynamics, elasticity, general relativity, and quantum mechanics.

Definition 2.1 *(Partial Differential Equation [16])*

Let us fix an integer $k > 1$ and let U denote an open subset of $\mathbb{R}^n$. An expression of the form

$$F\left(x, u, \frac{\partial u}{\partial x}, \ldots, \frac{\partial^{k-1} u}{\partial x^{k-1}}, \frac{\partial^k u}{\partial x^k}\right) = 0 \qquad (x \in U) \tag{2.1}$$

is called a k^{th}-order partial differential equation, where

$$F : \mathbb{R}^{n^k} \times \mathbb{R}^{n^{k-1}} \times \ldots \mathbb{R}^n \times \mathbb{R} \times U \to \mathbb{R} \tag{2.2}$$

is given, and $u : U \to \mathbb{R}$ is the unknown function. Here, $\mathbb{R}^{n^k}$ represents the space of all k-order partial derivatives.

PDEs that describe the evolution of a system that depends on a continuous time variable t are called *evolutionary PDEs*. An evolutionary PDE can be written in the following abstract form [11]:

$$\frac{\partial u}{\partial t} = A(u), \;\; u|_{t=0} = u_0(x). \tag{2.3}$$

Here $u = u(x, t)$ represents the solution of Eq. (2.3), and x, t denote the spatial and time variables, respectively.

2.2 Quadratization Algorithm Definitions

2.2.1 Notation

Throughout this work, we will avoid the explicit notation of functions in the PDE systems for ease of notation; thus, we denote a function depending on some independent variables by only its symbolic name, for example, $u = u(t, x)$. Also, we will use the subscript notation of partial derivatives, e.g., $\dfrac{\partial u}{\partial x} = u_x$, $\dfrac{\partial^2 u}{\partial x^2} = u_{xx}$, and so on.

The sets of polynomials pertinent to this work are defined by $\mathbb{R}[\mathbf{U}]$, where $\mathbf{U}$ is the set of indeterminates formed by the unknown functions and their partial derivatives in a PDE

system. A product of positive-integer powers of variables is referred to as *monomial* (e.g., $u^5 u_x$), and the total degree of a monomial is the sum of the powers of the variables appearing in it. A *polynomial* is a sum of monomials, e.g., $u + u_x^2$. Let p be a polynomial, then $\deg(p)$ denotes the total degree of p, the maximum of the total degrees of the monomials appearing in p. For example $\deg_u(u^5 u_{xx}) = 5$ and $\deg(u^5 u_{xx}) = 6$.

Furthermore, given an ordering of monomials, the term that ranks the highest within a polynomial is called its leading term. The ordering we will use throughout this work orders power products first by their degree and then lexicographically.

2.2.2 Quadratization for PDEs

A quadratization for PDEs is a transformation of a system of PDEs with a polynomial right-hand side into a system of PDEs with at most quadratic right-hand side via introducing new variables. The next definition is based on the work of Pogudin and Bychkov [10, Definition 1], modified for the PDE case.

Definition 2.2 *(Quadratization for PDE systems)*

Consider a polynomial system of one-dimensional time-dependent PDEs of the form

$$u_{1_t} = p_1(\boldsymbol{U}), \ldots, u_{n_t} = p_n(\boldsymbol{U}), \tag{2.4}$$

with $p_1, \ldots, p_n \in \mathbb{R}[\boldsymbol{U}]$, and $u_1, \ldots, u_n$ the unknown functions of the system. Now, we define a set $\boldsymbol{W}$ composed of the new variables

$$w_1 = g_1(\boldsymbol{U}), \ldots, w_l = g_l(\boldsymbol{U}), \tag{2.5}$$

with $g_1, \ldots, g_l$ functions of the set $\boldsymbol{U}$. Let us call $\boldsymbol{D}^k_{\boldsymbol{W}}$ the set of partial derivatives of each of the variables in $\boldsymbol{W}$ up to an order k. Then, $\boldsymbol{W}$ is said to be a quadratization of Eq. (2.4) if there exist an integer k and polynomials $q_1(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{D}^k_{\boldsymbol{W}}), \ldots, q_{n+l}(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{D}^k_{\boldsymbol{W}})$ with total degree at most two such that

$$u_{1_t} = q_1(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{D}^k_{\boldsymbol{W}}), \ldots, u_{n_t} = q_n(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{D}^k_{\boldsymbol{W}}) \text{ and} \tag{2.6}$$

$$w_{1_t} = q_{n+1}(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{D}^k_{\boldsymbol{W}}) \ldots w_{l_t} = q_l(\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{D}^k_{\boldsymbol{W}}), \tag{2.7}$$

The size of the set $\mathbf{W}$ is called *the order of quadratization*. A quadratization of the smallest possible order is called an *optimal quadratization*. If all functions $g_1(\mathbf{U}), \ldots, g_l(\mathbf{U})$ are monomials, the quadratization is called a *monomial quadratization*. If a monomial quadratization has the smallest possible order among all the monomial quadratizations for the system, it is called *optimal monomial quadratization*.

Moreover, recent unpublished work proves that a monomial quadratization exists for every PDE system [38]. Additionally, according to Hemery et al. [24], the optimization problem

of determining the minimum number of variables necessary to define an equivalent quadratic polynomial ordinary differential equation system is NP-Hard. Since we can view ODEs as particular cases of PDEs, we expect that the same problem is NP-Hard for the entire class of PDEs.

2.2.3 Gaussian Elimination Algorithm

Gaussian elimination (GE) [25] is the standard method for solving a system of linear equations. As such, it is one of the most ubiquitous numerical algorithms and plays a fundamental role in scientific computation. Using elementary row operations, GE aims to reduce a full system of n linear equations in n unknowns to triangular form.

A matrix is in reduced row echelon form (RREF) [2] (for square matrices, triangular form) if and only if all the following conditions hold:

- The first nonzero entry in each row is 1.
- Each successive row has its first nonzero entry in a later column.
- All entries (above and) below the first nonzero entry of each row are zero.
- All full rows of zeroes are the final rows of the matrix.

The idea behind Gaussian elimination [19, Section 3.2] is to convert a given system

$$Ax = b \tag{2.8}$$

to an equivalent triangular system. This conversion is achieved by taking appropriate linear combinations of the equations. In the language of matrix factorizations, the idea [43, Lecture 20] is to transform A into an upper-triangular matrix U by introducing zeros below the diagonal for every column. This "elimination" process is equivalent to multiplying A by a sequence of lower-triangular matrices L_k on the left:

$$\underbrace{L_{m-1} \dots L_2 L_1}_{L^{-1}} A = U. \tag{2.9}$$

Setting $L = L_1^{-1} L_2^{-1} \dots L_{m-1}^{-1}$ gives

$$A = LU. \tag{2.10}$$

LU is a factorization of A, where U is upper-triangular and L is lower-triangular. For example, if we have a 4×4 matrix, the algorithm follows the three steps shown in Figure 2.1.

The kth transformation L_k introduces zeros below the diagonal in column k by subtracting multiples of row k from rows $k+1, \dots, m$. Since the first $k-1$ entries of row k are already zero, this operation does not destroy any zeros previously introduced. The algorithm shown in Algorithm 1 is one of several formulations of GE for a $A \in \mathbb{R}^{m \times m}$ matrix.

The Gauss-Jordan algorithm [15] is equivalent to GE, followed by a further reduction of the resulting upper triangular system to a diagonal system. Given the system in Eq. (2.8),

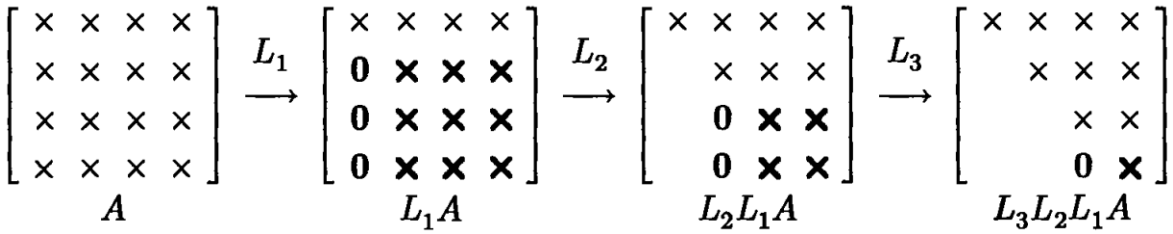

Figure 2.1: Figure from [43] to show LU factorization. Boldfacing indicates entries just operated upon, and blank entries are zero.

Algorithm 1 Gaussian Elimination [43]

```
U ← A
L ← I
for k ∈ {1, ..., m − 1} do
    for j ∈ {k + 1, ..., m} do
        l_jk = u_jk / u_kk
        u_{j,k:m} = u_{j,k:m} − l_jk u_{k,k:m}
    end for
end for
```

the application of the Gauss-Jordan elimination algorithm is equivalent to performing n successive transformations, starting from the original matrix $A^{(1)} = A$ and right-hand side $b^{(1)} = b$. The overall effect is the transformation of $A^{(1)}$ with permuted columns into the identity matrix.

2.2.4 Depth-first Search

Depth-first search (DFS) [41] is a technique that has been widely used for finding solutions to problems in combinatorial theory, algorithms on trees and graphs, and artificial intelligence. Given a graph G we want to explore, there are many ways of searching through the vertices of G. Consider the following selection rule: when choosing an edge to traverse, always choose an edge starting from the last vertex reached that still has unexplored edges. A search that uses this rule is called a *depth-first* search.

If n is the number of vertices and m is the number of edges, DFS takes time $O(m + n)$ [29], since each edge is stacked at most once in each direction, and each edge and vertex requires a constant amount of processing.

2.2.5 Best-first Search

Of all search strategies used in problem-solving, one of the most popular methods of exploiting heuristic information to cut down search time is the informed best-first strategy [14]. The general philosophy of this strategy is to use the heuristic information to assess the "merit" latent in every candidate exposed during the search and then continue the exploration along

the direction of the highest merit.

Given a weighted directional graph G with a start node s and a set of goal nodes Γ, the optimal path problem is to find a least-cost path from s to any member of Γ where the cost of the path may, in general, be an arbitrary function of the weights assigned to the nodes and branches along that path. A general best-first (GBF) strategy will pursue this problem by constructing a tree T of selected paths of G using the elementary operation of node expansion, that is, generating all successors of a given node. Starting with s, GBF will select for expansion that leaf node of T with the highest "merit" and will maintain in T all previously encountered paths that still appear viable candidates for sprouting an optimal solution path. The search terminates when no such candidate is available for further expansion, in which case the best solution path found so far is issued as a solution; if none has been found, a failure is declared.

2.2.6 Branch-and-Bound framework

The branch-and-bound (B&B) [36] framework is a fundamental and widely used methodology for producing exact solutions to NP-hard optimization problems. B&B encapsulates a family of algorithms that share a common core solution procedure. This procedure implicitly enumerates all possible solutions to the problem under consideration by storing partial solutions called subproblems in a tree structure. Unexplored nodes in the tree generate children by partitioning the solution space into smaller regions that can be solved recursively (i.e., branching), and rules are used to prune off regions of the search space that are suboptimal (i.e., bounding). Once the entire tree has been explored, the best solution found in the search is returned.

Furthermore, three components can have a high impact on the efficiency of finding a solution [36]:

- Search strategy: The search strategy in a B&B algorithm determines the order in which unexplored subproblems in the built tree are selected for exploration. The choice of search strategy has potentially significant consequences for the computation time required for the B&B procedure, as well as the amount of memory used.
- Branching strategy: This component determines how the solution space is partitioned to produce new subproblems in the tree. Branching strategies can be categorized into two groups: binary branching strategies and non-binary or wide branching strategies.
- Pruning rules: These are rules that prevent exploration of suboptimal regions of the tree. It is important to highlight that any node that cannot be pruned by the pruning rules must be explored by any search strategy. The only way to reduce the size of the search tree in this case is to use better pruning rules. Thus, this component is a critical aspect of the B&B framework.

To give an overview of the instructions that a B&B algorithm follows, we first define some elements of the problem. The optimization problem we are trying to solve will be

defined by $\mathcal{P} = (X, f)$, where X is the search space (set of valid solutions to the problem) and $f : X \to \mathbb{R}$ is the objective function. Then, we call $x^* \in \arg\min_{x \in X} f(x)$ the optimal solution we are trying to find. The search tree that the algorithm builds iteratively is T, and $\hat{x} \in X$ is a feasible solution that will be stored globally (best solution yet). In each iteration, the algorithm selects a new subproblem $S \subseteq X$ from a list L of unexplored subproblems. With these notions, we show a pseudocode for the generic B&B procedure in Algorithm 2 that describes the instructions using the elements introduced above.

Algorithm 2 Generic Branch & Bound(X, f) [36, Section 2.1]

```
L ← {X}
Initialize x̂
while L ≠ ∅ do
    Select a subproblem S from L to explore
    if a solution x̂' ∈ {x ∈ S‖f(x) < f(x̂)} is found then
        x̂ ← x̂'
    end if
    if S cannot be pruned then
        Partition S in S_1, S_2, ... S_r
        Insert S_1, S_2, ... S_r into L
    end if
    Remove S from L
end while
    return x̂
```

A short example of how the algorithm and the components of a B&B approach work is to solve the classic Traveling Salesman Problem (TSP). In TSP, given a set of cities, the goal is to find the shortest possible route that visits each city exactly once and returns to the original city. Let us consider four cities A, B, C, D, and the distance matrix in Table 2.1 representing the distances between each pair of cities:

Table 2.1: Distance matrix for TSP B&B example

from \ *to*	A	B	C	D
A	0	4	7	6
B	8	0	14	9
C	7	9	0	11
D	16	6	8	0

Let us start by obtaining a candidate route using a greedy approach. We begin with an arbitrary city, in this case D, and choose the closest city (in our example B) to continue the tour. In every step, we always pick the nearest city that has not yet been selected until we return to the initial city. Then our candidate route is $D \to B \to A \to C \to D$ of cost 32.

To establish a bounding criteria that provides an underestimate of the cost to complete a tour from any city, we define the heuristic:

$$h(x, y) = min_{route} + (cost(y) - min_{local}(x)).$$

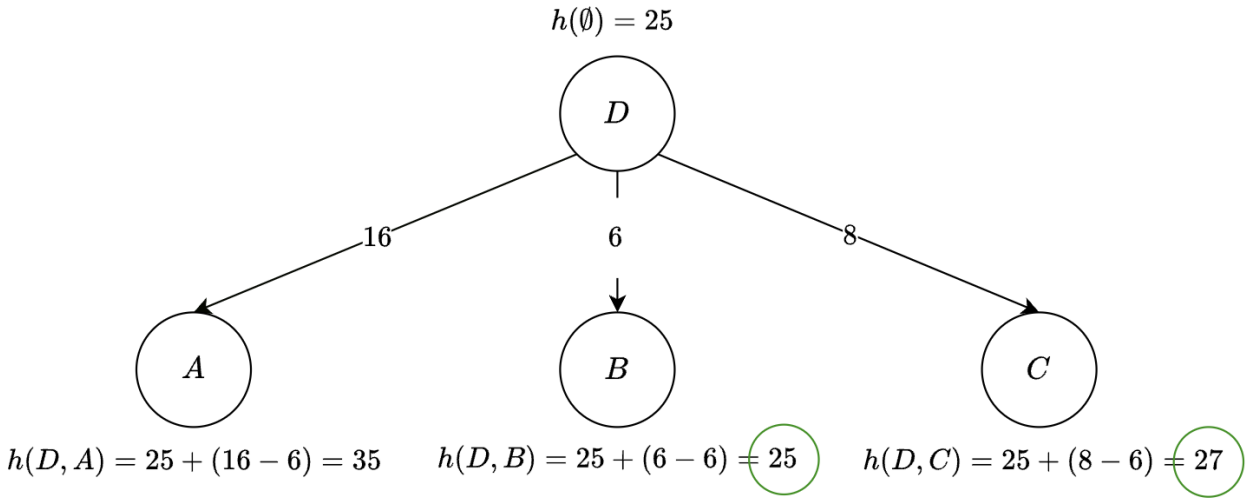

Figure 2.2: First step of B&B algorithm for TSP example

Where x is the last city visited, y a prospective new city to be selected, $cost(y)$ is the cost it would take to visit city y, and $min_{local}(x)$ is the minimum cost from city x to any other. On the other hand, min_{route} relates to the minimal cost we could obtain from taking this specific route, which is mathematically the result of the heuristic h on the parent node x. At the beginning of the tour, we define min_{route} as the minimal cost in every row of the matrix in Table 2.1: $min_{route}(\emptyset) = 4 + 8 + 7 + 6 = 25$, so we know the overall minimal route is no lesser than 25.

We start again from city D and identify its min_{local} as 6, which is the cost to B, the nearest city from D. Then, we calculate the heuristic $h(D, x)$ for every node in the second level of the tree. We show the results in Figure 2.2.

Since we already have a candidate tour of cost 32, we can prune all branches in level two with heuristic results greater than 32. In our case, we discard the first branch from left to right in Figure 2.2 and decide to explore the best option so far: visiting the city B. From city B, we expand our possibilities and calculate heuristic h for each remaining city as shown in Figure 2.3.

As we can see in Figure 2.3, at the third level we cannot prune any branch, so we explore solutions in a best-first manner. Then, we choose city A and obtain the final route $D \to B \to A \to C \to D$, which gives us the total cost $6 + 8 + 7 + 11 = 32$. After we obtain this result, we perform backtracking until the first level where we encounter unpruned nodes as children, in this case, the node of city B in the second level of the tree. Then, we explore the remaining nodes.

We keep performing the steps described above until each branch that is not pruned is explored. We show the remaining steps of the algorithm in Figure 2.4. Finally, the algorithm chooses the path $D \to C \to A \to B \to D$ as the shortest possible route.

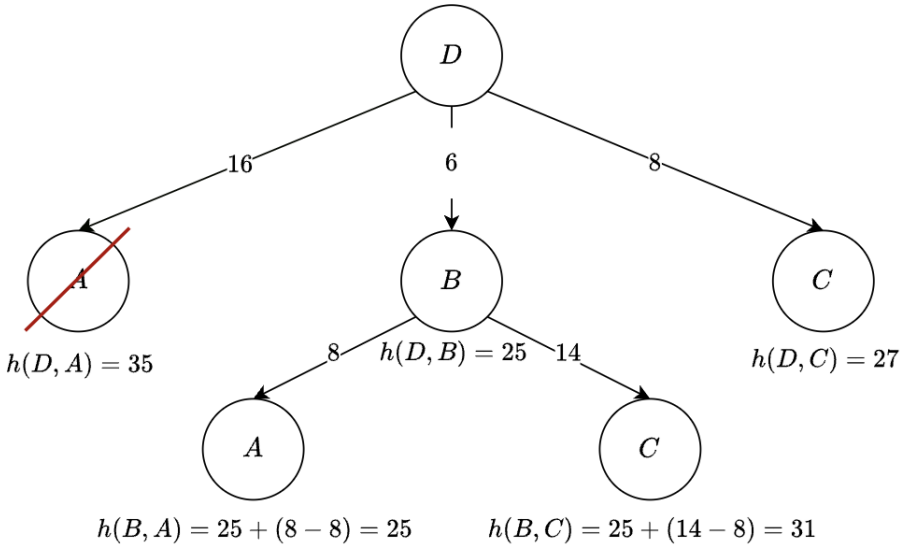

Figure 2.3: Second step of B&B algorithm for TSP example

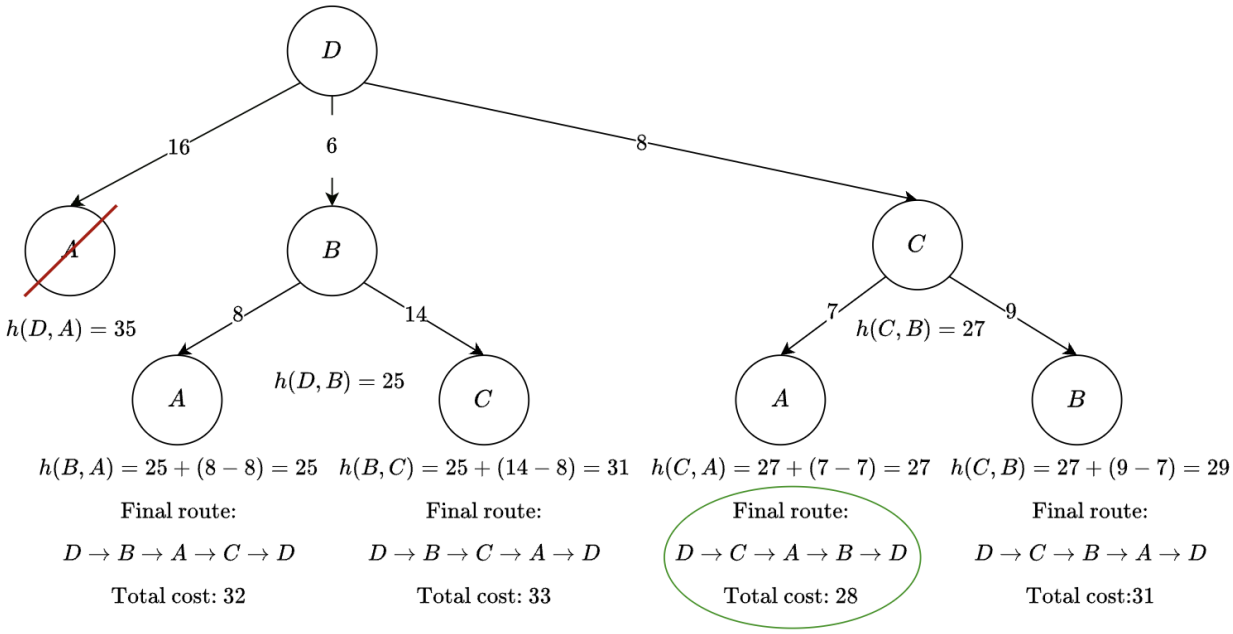

Figure 2.4: Final steps of B&B algorithm for TSP example

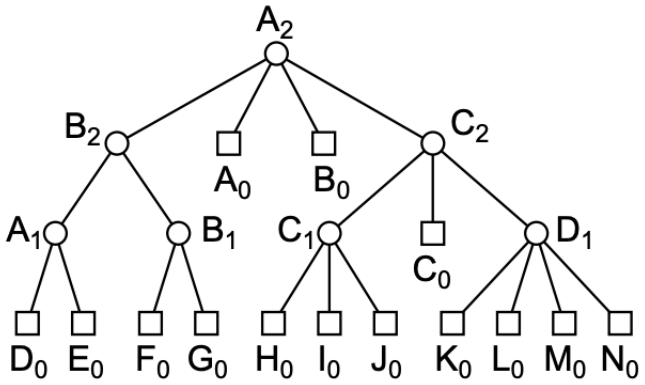

Figure 2.5: An example of a search hierarchy [26].

2.2.7 Incremental nearest neighbor algorithm

The incremental nearest neighbor algorithm [26] aims to solve proximity search problems. It applies whenever the search space can be represented hierarchically, provided certain conditions hold. The algorithm operates on a finite set $S \subset \mathbb{U}$ of objects, a query object $q \in \mathbb{U}$, and a data structure T that organizes the set S or provides information about it. The search hierarchy is composed of elements e_t of several different types t, with $t = 0, \ldots, t_{max}$.

As depicted in Figure 2.5, the search hierarchy forms a tree with the objects as leaves. Each node represents a subset of S, with an element e_0 of type 0 representing a single object in S. An element e_t of type t can give rise to one or more "child" elements of types 0 through t_{max}. Thus, the search problem for e_t is decomposed into several smaller subproblems. Also, elements of each type t have an associated distance function $d_t(q, e_t)$ for measuring the distance from a query object q to the elements of that type, where $d_t(q, e_t) \leq d_{t'}(q, e_{t'})$ for any element $e_{t'}$ that is a descendant of e_t in the search hierarchy.

This algorithm traverses the search hierarchy using a best-first strategy (Section 2.2.5), similar to the A*-algorithm [12]. At any step of the algorithm, the algorithm visits the element with the smallest distance from the query object among all unvisited elements whose parents have been visited. This is done by maintaining a global list of elements organized by their distance from the query object. A data structure that supports the necessary operations (insert and delete) is an instance of a priority queue. To break ties among elements having the same distance, the elements with lower type numbers have priority, and among elements of the same type, priority is given to elements deeper in the search hierarchy. The general incremental nearest neighbor algorithm is shown in Algorithm 3.

Algorithm 3 Generalized incremental nearest neighbor algorithm [26, Section 2.3]

```
Queue ← NewPriorityQueue()
e_t ← root of the search hierarchy induced by q, S and T
Enqueue(Queue, e_t, 0)
while not IsEmpty(Queue) do
    e_t ← Dequeue(Queue)
    if t = 0 then
        Report e_t as the next nearest object
    else
        for each child element e'_t of e_t do
            Enqueue(Queue, e'_t, d_t'(q, e_t'))
        end for
    end if
end while
```

2.2.8 Gröbner Bases

The theory of Gröbner bases [7] provides a uniform approach to solving a wide range of problems expressed in terms of sets of multivariate polynomials. Moreover, many problems in different areas of mathematics can be reduced to the problem of computing Gröbner bases. To define this method clearly, let us first review the division or reduction of multivariate polynomials. To do this, we introduce the following example polynomials:

$$g = x^2y^3 + 3xy^2 - 5x, \tag{2.11}$$

$$f_1 = xy - 2y, \quad f_2 = 2y^2 - x^2, \tag{2.12}$$

$$F = \{f_1, f_2\}, \tag{2.13}$$

where g, f_1, f_2 are bivariate polynomials and F is a polynomial set. The monomials in these polynomials follow an ordering. These orderings can be infinite, with the most important ones being the lexicographic orderings and the orderings that, first, order power products by their degree and then lexicographically. In the example above, the monomials are ordered lexicographically with y ranking higher than x and are presented in descending order from left to right.

In this setting, we could perform a reduction for g by doing the following:

$$h = g - 3(y)f_1 = -5x + 6y^2 + x^2y^3. \tag{2.14}$$

By subtracting a suitable monomial multiple of f_1 from g, one of the monomials of g is canceled against the leading term of $-(3y)f_1$. To explain this situation, we say that "g is reduced to h modulo f_1" and write:

$$g \xrightarrow{f_1} h. \tag{2.15}$$

Given a set F of polynomials and a polynomial g, many different reductions of g modulo

polynomials in F may be possible. For example, for g and F in Eq. (2.11) and Eq. (2.13), we also have

$$h_2 = g - (\frac{1}{2}x^2y)f_2 = -5x + \frac{x^4y}{2} + 3xy^2, \tag{2.16}$$

and, hence,

$$g \xrightarrow{f_2} h_2. \tag{2.17}$$

Moreover, if for some polynomial f in F $g \xrightarrow{f} h$ holds we write

$$g \xrightarrow{F} h, \tag{2.18}$$

and if g reduces to h by finitely many reduction steps with respect to F the notation is

$$g \xrightarrow{F^*} h. \tag{2.19}$$

Furthermore, we write h_F if h cannot be reduced further with respect to F.

Now, we introduce the definition of Gröbner bases:

Definition 2.3 *(Gröbner bases)*

$$F \textit{ is a Gröbner basis} \Longleftrightarrow \underset{g,h,k}{\forall}(g \xrightarrow{F^*} h_F \wedge g \xrightarrow{F^*} k_F \Longrightarrow h = k) \tag{2.20}$$

In other words, this definition states that F is a Gröbner basis if and only if $g \xrightarrow{F^*}$ is unique.

2.2.9 Buchberger's algorithm

Buchberger's algorithm is a method for transforming a given set of polynomials into a Gröbner basis. To describe the steps of this algorithm in detail, let us introduce some definitions.

Definition 2.4 *(Ideals) [13]*

A subset $I \subset k[x_1, \ldots, x_n]$ is an ideal if it satisfies:

- $0 \in I$
- *If $f, g \in I$, then $f + g \in I$*
- *If $f \in I$ and $h \in k[x_1, \ldots, x_n]$, then $hf \in I$*

Definition 2.5 *(Polynomial ideals) [13]*

Let $f_1, \ldots, f_s$ be polynomials in $k[x_1, \ldots, x_n]$. Then we set

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i : h_1, \ldots, h_s \in k[x_1, \ldots, x_n] \right\}. \tag{2.21}$$

Here $\langle f_1, \ldots, f_s \rangle$ is an ideal.

On the other hand, the S-polynomial of two polynomials f_1 and f_2 refers to the result of applying multiplication and subtraction between them to cancel their leading terms. For example, if we consider the functions f_1 and f_2 defined in Eq. (2.12), then their S-polynomial is:

$$S - polynomial[f_1, f_2] = y f_1 - \frac{1}{2} x f_2 = \frac{x^3}{2} - 2y^2. \tag{2.22}$$

The computation of the $S-polynomial$ of two given polynomials involves multiplying the two polynomials by such monomial factor that the leading power product of both polynomials becomes equal, namely the least common multiple of the leading power products of the two polynomials.

Now, given a finite set F of multivariate polynomials, Buchberger's algorithm's main goal is to find a set of polynomials G such that $Ideal(F) = Ideal(G)$ and G is a Gröbner basis. The description of this algorithm is shown in Algorithm 4.

Algorithm 4 Buchberger's algorithm [7, Section 6]

```
G ← F
for any pair of polynomials f1, f2, ∈ G do
    g ← S-polynomial of f1, f2
    reduce g to a reduced form h with respect to G
    if h = 0 then
        consider the next pair
    else
        add h to G and iterate
    end if
end for
```

2.3 Related Work

2.3.1 State of the art

Optimal monomial quadratization for ODE systems [10]

In this work, quadratization is handled as a transformation of a system of ODEs with a polynomial right-hand side into a system of ODEs with at most quadratic right-hand side via introducing new variables.

This paper presents an algorithm that, given a system of polynomial ODEs, finds a transformation into a quadratic ODE system by introducing new variables that are monomials in the original variables. This algorithm is guaranteed to produce an optimal transformation of this form for the ODE case (that is, the number of new variables is as small as possible).

This algorithm follows the general Branch-and-Bound (B&B) paradigm. To describe the instructions followed by the designed algorithm, the next definitions are introduced:

Definition 2.6 *(B&B formulation for the ODE quadratization problem)*

- *The search space is a set of all monomial quadratizations of the input system* $\bar{x}' = \bar{f}(\bar{x})$.
- *The objective function to be minimized is the number of new variables introduced by a quadratization.*
- *Each subproblem is defined by a set of new monomial variables* $z_1(\bar{x}), \ldots, z_l(\bar{x})$ *and the corresponding subset of the search space is the set of all quadratizations including the variables* $z_1(\bar{x}), \ldots, z_l(\bar{x})$.

Definition 2.7 *(Properties of a subproblem in the ODE quadratization problem)*

To each subproblem (see Definition 1) defined by new variables $z_1(\bar{x}), \ldots, z_l(\bar{x})$, *we assign:*

1. *the set of generalized variables, denoted by V, consisting of the polynomials* $1, x_1, \ldots, x_n$, *and* $z_1(\bar{x}), \ldots, z_l(\bar{x})$;
2. *the set of nonsquares, denoted by NS, consisting of all the monomials in the derivatives of the generalized variables that do not belong to* $V^2 := \{v_1 v_2 | v_1, v_2 \in V\}$. *In particular, a subproblem is a quadratization iff* $NS = \emptyset$.

The two previously presented definitions describe the context of implementing the Branch-and-Bound paradigm, which can also be applied to the quadratization of PDEs. This algorithm was developed in Python and it is stated that this implementation compares favorably with the existing software and finds better quadratizations for already used benchmark problems. However, it is important to note that this algorithm guarantees optimal quadratization only for ODE systems. While it is possible to apply this method to PDE systems (as described in the following section), optimality does not hold within the realm of PDEs.

Dimension-agnostic quadratization for semi-discretized PDEs [9]

This paper presents theory, algorithms, and software capabilities for quadratizing non-autonomous ODEs. Here, an algorithm was designed and implemented that generalizes the process of quadratization for systems with arbitrary dimensions that retain the nonlinear structure when the dimension grows. Such systems were provided with a dimension-agnostic quadratization. An example of these systems is semi-discretized PDEs, where the nonlinear terms remain symbolically identical when the discretization size increases.

The algorithm mentioned above is an extension from the previous version of *QBee* (algorithm for quadratizing ODEs [10]), including extra functionality to (i) optimally quadratize polynomial systems with time-dependent inputs, (ii) to find dimension-agnostic quadratizations for systems with variable dimension (e.g., semi-discretized PDEs).

Exploring more into this new quadratization method that applies to families of ODE systems of variable dimension, for which it produces a dimension-agnostic quadratization, the most natural use-cases of this new method are ODEs that are derived via semi-discretization of PDEs, i.e., where the symbolic form of the nonlinear terms stays the same, but the discretization dimension n of the system can be varied. According to this work, the definition of dimension-agnostic quadratization is as follows:

Theorem 2.8 *(Dimension-agnostic quadratization)*

Consider a family of linearly coupled ODEs defined by $n_d + 1$ polynomial vectors $\boldsymbol{p}_0(\boldsymbol{x})$, $\ldots, \boldsymbol{p}_{n_d}(\boldsymbol{x})$ in $\boldsymbol{x} = [x_1, \ldots, x_{n_d}]^\top$, and consider a l-dimensional vector $\boldsymbol{w}_1(\boldsymbol{x}) \in \mathbb{C}[\boldsymbol{x}]^l$ together with an L-dimensional vector $\boldsymbol{w}_2(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \in \mathbb{C}[\boldsymbol{x}, \tilde{\boldsymbol{x}}]^L$, where $\tilde{\boldsymbol{x}} = [\tilde{x}_1, \ldots, \tilde{x}_{n_d}]^\top$ are formal variables used as placeholders for the coupled variables (to be made precise below). For every integer n and matrices $\boldsymbol{D}_1, \ldots, \boldsymbol{D}_{n_d} \in \mathbb{C}^{n \times n}$, we define a set

$$
\begin{aligned}
M(\boldsymbol{w}_1, \boldsymbol{w}_2; \boldsymbol{D}_1, \ldots, \boldsymbol{D}_{n_d}) := &\{\boldsymbol{w}_1(\boldsymbol{x}_i^{[n]}) | 1 \leq i \leq n\} \quad \cup \\
&\{\boldsymbol{w}_2(\boldsymbol{x}_{i_0}^{[n]}, \boldsymbol{x}_{i_1}^{[n]}) | 1 \leq i_0 \neq i_1 \leq n, \quad \textit{and} \quad \exists k : (\boldsymbol{D}_k)_{i_0, i_1} \neq 0\}.
\end{aligned}
$$

We then say that $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are a dimension-agnostic quadratization of the family if, for every integer n and matrices $\boldsymbol{D}_1, \ldots, \boldsymbol{D}_{n_d} \in \mathbb{C}^{n \times n}$, the set $M(\boldsymbol{w}_1, \boldsymbol{w}_2; \boldsymbol{D}_1, \ldots, \boldsymbol{D}_{n_d})$ is a quadratization of the system $F^{[n]}(\boldsymbol{p}_0(\boldsymbol{x}), \ldots, \boldsymbol{p}_{n_d}(\boldsymbol{x}), \boldsymbol{D}_1, \ldots, \boldsymbol{D}_{n_d})$.

With this definition, the extension for tackling dimension-agnostic quadratizations is shown in Algorithm 5.3 of Section 5 from the work of Bychkov et al. [9]. In Section 6 of this work, there are some results of quadratization on PDE discretized models (Sections 6.1 and 7), obtaining quadratizations of four new variables for both models.

It is worth noting that the method described above can indeed computationally quadratize a PDE system. However, it is important to emphasize that this requires discretizing the system as a preprocessing step, leading, in many cases, to a large-scale system of ODEs. Since the number of state variables of such a system easily might exceed dimensions up to $\mathcal{O}(10^5)$ [4], a fast and reliable simulation is hardly possible, reducing the procedure's practicality in certain applications.

Chapter 3

QuPed Foundations

We designed and implemented an algorithm named *QuPed* to address the PDE quadratization problem. The description of how it works will be divided into three parts. The first one describes how the algorithm decides which sets of variables are candidates for a quadratization, and how it searches for the optimal one through two different design paradigms: *Branch-and-Bound (B&B)* and *Nearest Neighbor* algorithm with best-first search principle; the second part shows how the algorithm verifies if a set of variables is a quadratization; and the last part describes a functionality added to the algorithm to handle rational functions within the right-hand side equations of a PDE, which are very common in systems that model engineering problems. Lastly, we discuss a short observation related to the success of our algorithm for finding a quadratization.

3.1 Finding Optimal Set of New Variables

To create a search space of possible quadratizations, we first need to describe how the algorithm proposes new variables from the equations of a given PDE. For this, the algorithm finds a group of sets (decompositions) for every monomial with a degree greater than two within the PDE's right-hand side equations. These decompositions are all the ways we can introduce new variables such that the original monomial in the equation will be at most quadratic (multiplication of at most two elements). For example, the decompositions for the monomial $u_x^2u^2$ are $\{u_x^2, u^2\}$, $\{1, u_x^2u^2\}$, $\{u_x^2u, u\}$ and $\{u_x, u_xu^2\}$. If we use the first decomposition to rewrite the original term, we need to introduce two new variables: u_x^2 and u^2.

Therefore, finding the decompositions described below for a determined monomial is equivalent to answering the combinatorial question: How many ways can we divide a set of elements of size n into two subsets? with n the degree of the monomial to decompose. Let us solve this combinatorial problem to obtain an upper bound for the number of decompositions of a monomial, which represent the potential new variables and our search space.

First, for each element within the set, we can decide whether to include it in the first subset or not. Thus, we have two options for each element, which gives us the expression:

$$2^n. \tag{3.1}$$

Nonetheless, we are counting twice the number of cases because the decomposition $\{x, y\}$ is the same as the decomposition $\{y, x\}$. Therefore, we divide the result in Eq. (3.1) by two:

$$\frac{2^n}{2} = 2^{n-1}. \tag{3.2}$$

The expression in Eq. (3.2) is an upper bound for each nonquadratic monomial in the system, as it assumes that every element within the monomial is distinguishable.

Now that we have stated how to obtain possible new variables and an upper bound for the number of them, we describe the instructions followed by the algorithm for finding and sorting candidate variables for a quadratization:

1. For every polynomial in the system that is not quadratic, store all the possible ways of decomposing every monomial so that its degree is equal to or lower than two. According to the result obtained in Eq. (3.2), this will be at most 2^{n-1} decompositions, where n is the maximum degree in the PDE system.
2. Sort this set of decompositions according to certain heuristics to get a ranked list of expressions. Then, we obtain a sorted list of variables based on whether they have a higher chance of being a quadratization or if they derive fewer decompositions. The heuristics implemented are listed below.
    - By order and degree: it sorts the resulting expressions of the decomposition by giving priority to expressions of lower order of derivatives and then of lower degree.
    - By degree and order: it sorts by lower degree and secondly by lower order.
    - By a function of order and degree: in this case, the expressions are sorted using as criteria a linear function of the degree and order of derivatives in the expressions.

An example of how this algorithm works is shown below.

Example Consider the PDE depending on the time t and spacial x variables:

$$u_t = u_x^2 u. \tag{3.3}$$

Decomposing the right-hand side of Eq. (3.3) results in the set $\{(u_x^2, u), (u_x^2 u, 1), (u_x u, u_x)\}$.

Now, we dispose of expressions that do not suppose new variables or integers (for example u, u_x): $\{(u_x^2), (u_x^2 u), (u_x u)\}$. Then we sort by degree and order to obtain the final set: $\{u_x^2, u_x u, u_x^2 u\}$.

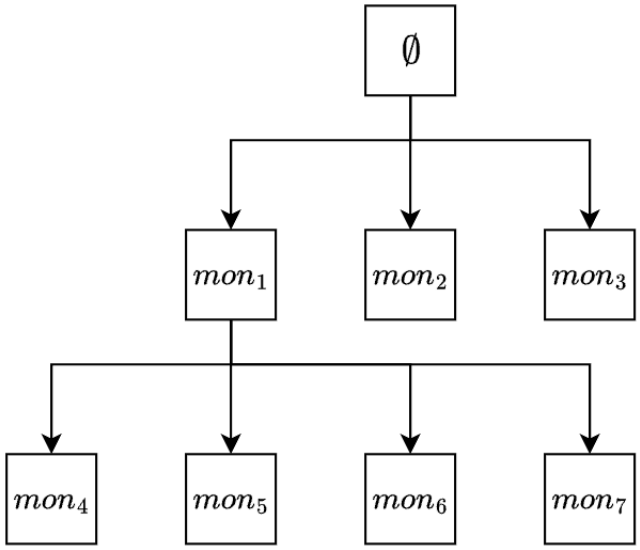

Figure 3.1: Hierarchy tree made of monomial decompositions

3.1.1 Branch-and-Bound Algorithm

Now, for finding an optimal quadratization of the system, i.e., an optimal set of new variables, a *Branch-and-Bound* (B&B) algorithm was implemented. This framework is often used to find a minimal solution subset efficiently, which is the same problem to be addressed in this case: to find the smallest set of variables that is a quadratization for a given system.

To perform this search, we organize our space search as a tree (see Figure 3.1), where the root is an empty set, and its children nodes are all the monomial decompositions coming from the original PDE. Each time we enter one node, its children will be the new subproblems derived from the re-written form of the PDE system after introducing the variables within the parent node. Depending on which node (decomposition) we enter, we will introduce at most two new variables in each level.

To be more precise, we define how our problem translates to the B&B framework.

Definition 3.1 *(B&B formulation for the PDE quadratization problem)*

- *The search space is the set of all monomial decompositions of the input PDE system and subproblems.*
- *The objective function to minimize is the number of new variables introduced by a quadratization.*
- *Each subproblem is defined by a set of new monomial variables* $w_1(\boldsymbol{U}), \ldots, w_l(\boldsymbol{U})$ *and the corresponding subset of the search space is the set of all quadratizations including the variables* $w_1(\boldsymbol{U}), \ldots, w_l(\boldsymbol{U})$.

We also state the components of search strategy, branching strategy, and pruning rules [36] for our problem.

- Search strategy: the search strategy we use is depth-first search with backtracking [41, Section 2] [36, Section 3.1]. The order in which the subproblems are arranged in a single level and, therefore, traversed, is given by the sorting criteria chosen for the set of potential variables.
- Branching strategy: the solution space is partitioned by all the possible new variables that could be introduced to get a quadratization, that is, the PDE system right-hand side monomial decompositions from the parent node (wide branching).
- Pruning rules: pruning rules are introduced to prevent the exploration of suboptimal regions of the solutions tree. The currently implemented pruning rules are:
  - The smallest order (minimal number of variables) of a quadratization found within the visited nodes, which is our objective function to minimize.
  - The order of derivatives within the new variables. Given the physical dynamics that PDEs represent, there are, in some cases, existence or regularity restrictions for the differentiation order of undefined functions. Therefore, if one branch introduces variables with higher-order derivatives than the ones allowed for the problem, the algorithm prunes it. For the implementation of this algorithm, we took a conservative approach, and by default, we prune every branch that introduces a derivative order higher than the one within the original equation, as the contextual dynamics are unknown.

Our algorithm first checks if the branch it just entered needs to be pruned according to the pruning rules implemented. After verifying that the current branch will not produce a suboptimal solution, it will verify if the set of variables proposed in the current node is a quadratization. If so, it will return the quadratization set and its order (minimal number of variables until then). If it does not identify a quadratization with the proposed set, it will produce the children of the current node (subproblems) and explore them in a DFS manner. Finally, after confirming it is optimal, it will return the best quadratization set.

Now, let us see an example to have a clear idea of how the algorithm works:

Example Consider the following PDE:

$$u_t = u_x^3 + u^3. \tag{3.4}$$

After decomposing and sorting by degree and order the obtained tuples, we draw the tree shown in Figure 3.2.

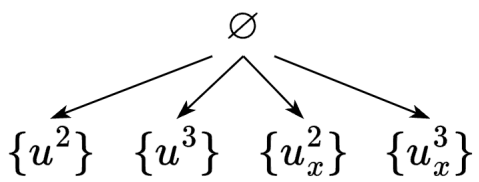

Figure 3.2: First level of variable tree from the PDE example in Eq. (3.4)

Now, the algorithm will enter the first branch from left to right, in this case, the system given by introducing the variable u^2. Once it adds this variable to the original PDE as w_0, we get the following system:

$$u_t = \frac{1}{2} \cdot u_x w_{0xx} - \frac{1}{2} \cdot w_{0x} u_{xx} + w_0 u, \tag{3.5}$$

$$w_{0_t} = 2u_x^3 u + 2w_0^2. \tag{3.6}$$

As we can see, it is not quadratized because $\deg(w_{0t})$ is greater than two. Therefore, we explore the subproblem of the branch selected, in this case, the rewritten form of the PDE shown in Eq (3.5) and Eq (3.6). After obtaining all the decompositions for the nonquadratic monomials (in this case, only $2u_x^3 u$) and sorting them by degree and order, the tree for this search takes the form depicted in Figure 3.3.

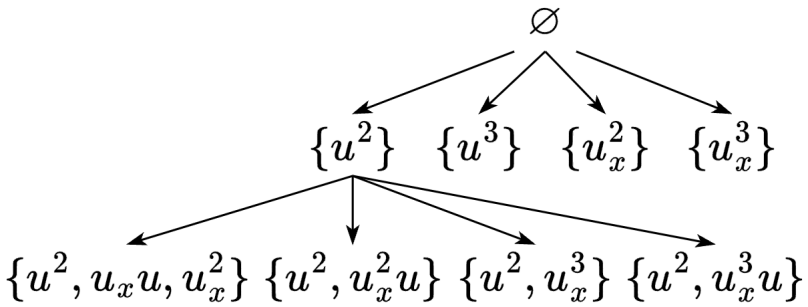

Figure 3.3: Second level of variable tree from Eq. (3.4)

When the algorithm checks for a quadratization in the branch given by $\{u^2, u_x u, u_x^2\}$, it finds that this set is indeed a quadratization. The expected behavior would be to check other branches with fewer variables in the hope of finding a lower-order quadratization. Essentially, to follow the path shown in Figure 3.4.

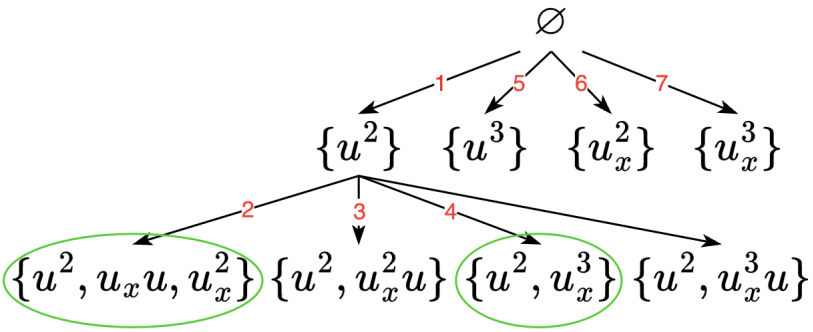

Figure 3.4: Full variable tree from Eq. (3.4) with quadratizations found in green and branches traversed in red.

Then, after the seventh node in Figure 3.4 is traversed, the algorithm should return the quadratization with the minimum order within the search space. The algorithm does not

explore the last children node from the second level as it identifies it will not obtain a better quadratization than the best one found. In the next subsection, we explain that with the introduction of a new function, this algorithm only traverses five nodes in total.

Shrink Function

This function was implemented so that every time the algorithm finds a quadratization, it will try to find a better one within all the subsets (*powerset*) of the quadratization set found. If it finds a better one and the number of variables introduced is smaller than the global one, it will use this new bound for the next branches, giving the algorithm one more tool for pruning branches quicker.

Now, let us examine the effect of this function using the previous example.

Example We consider again the PDE $u_t = u_x^3 + u^3$. Then, the variable tree for finding the optimal quadratization takes the form shown in Figure 3.5 when the algorithm finds the first quadratization.

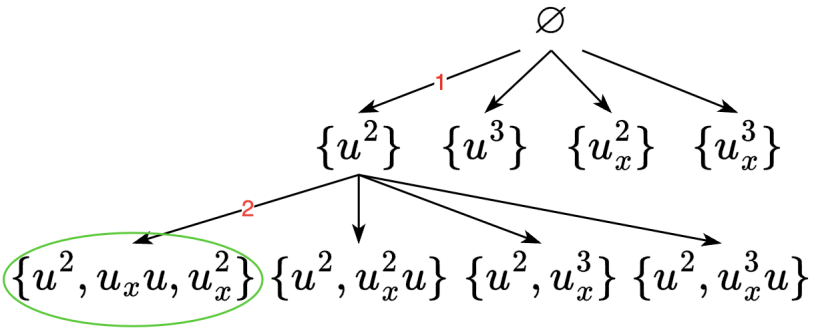

Figure 3.5: State of variable tree from Eq. (3.4) when the first quadratization is found

Instead of searching for smaller quadratizations in other branches, the algorithm will verify for each subset of the quadratization found if it is a quadratization for the system. In this case, it will look for a quadratization among the sets $\{u^2\}, \{u_x^2\}, \{u_x u\}, \{u^2, u_x^2\}$. In the last one, it identifies a quadratization; then, it stops searching within the rest of the node subsets and updates the minimum quadratization order found, i.e., the criteria for one of the pruning rules. Given the pruning rule update, the algorithm's search through the tree is the one described in Figure 3.6.

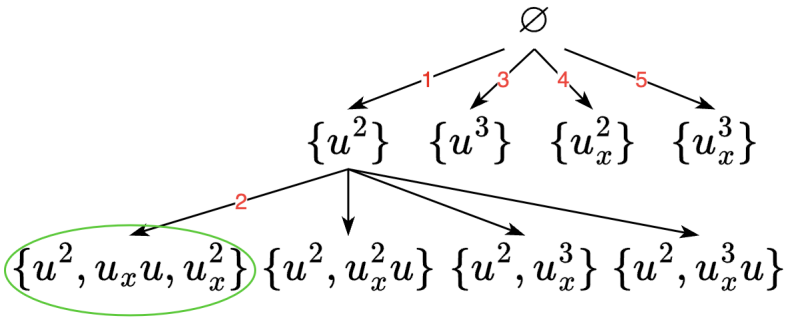

Figure 3.6: Search made by the algorithm in the variable tree

After traversing the fifth node, the algorithm returns the quadratization $\{u^2, u_x^2\}$.

3.1.2 Nearest Neighbor Algorithm

Another approach was implemented to search for an optimal quadratization. It follows the idea proposed by Hjaltason and Samet [26], in which they develop an algorithm called incremental nearest neighbor algorithm, suited for search spaces that can be represented hierarchically. This is the case of our problem, as seen in Figure 3.1, where the criteria for constructing the hierarchy of the nodes is the number of variables introduced for a quadratization.

To implement this algorithm, we define a finite set S of new possible variables, a query object q, and a tree as the data structure for organizing the elements in S. In this case, q will represent the original system without any new variable introduced. Our search hierarchy will be composed of the elements $e_0, \ldots, e_m$ with $m \leq 2^{|S|}$, which represent subsets of S, in our case, potential quadratizations where $e_0 = \emptyset$. Then, we introduce a distance function $d(q, e_i)$ with $i \in 0, \ldots, m$ for measuring the distance from the query object q to an element e_i, this distance represents how farther away is a quadratization from the original PDE system in terms of variables introduced (the lower the order of the quadratization, the closer it is from the original PDE). In our case, the distance function obeys the condition $d(q, e_i) < d(e_j)$ for any e_j descendant of e_i.

Essentially, this algorithm traverses the search hierarchy in a best-first manner instead of the more traditional depth-first or breadth-first traversals. More specifically, in every iteration, the algorithm visits the element with the smallest distance from q among all unvisited elements whose parents have been visited. To ensure this, we introduce a global priority queue, where the priority of an element e_i is the function $d(q, e_i)$. To break ties among elements having the same distance (same number of variables introduced), we use one of the sorting functions detailed in Section 3.1. Then, when a quadratization is found, it executes the shrink function to see if it can find a better quadratization. We show the pseudocode of the algorithm implemented.

Algorithm 5 Nearest Neighbor

```
PQ ← New Priority Queue
enqueue(PQ, e_0, 0)
while not isEmpty(PQ) do
    e_i ← dequeue(PQ)
    if e_i is a quadratization then
        powerset ← all subsets of e_i sorted by cardinality
        for subset ∈ powerset do
            if subset is a quadratization then return subset
            else
                return e_i
            end if
        end for
    else
        E ← subproblems from e_i
        for e_k ∈ E do
            enqueue(PQ, e_k, d(q, e_k))
        end for
    end if
end while
```

Furthermore, we also implemented an improvement for this algorithm. In this sense, we tried to control the size of the global priority queue by first storing the subproblems encountered in a separate FIFO queue instead of the global priority queue. Once the priority queue is almost empty, we extract a subproblem from the FIFO queue and insert its children nodes into the global priority queue. This improvement saves computational costs by reducing the number of times the insertion operation is performed in the priority queue data structure.

Let us use an example to show more clearly the instructions that this search algorithm follows for a specific PDE.

Example Consider the PDE dependent on time and space:

$$u_t = u_x^4. \tag{3.7}$$

At the start of the search for an optimal quadratization, the priority queue is empty. When the algorithm verifies that Eq. (3.7) is not quadratic, the possible sets of new variables for a quadratization are enqueued. In this case, the sorting function we use is by degree and order. Therefore, the scoring function $d(q, e_k)$ prioritizes sets with fewer variables, less degree, and then less order.

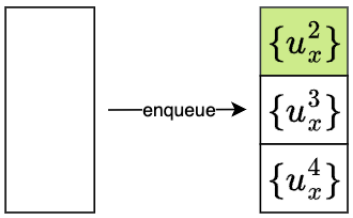

Figure 3.7: State of the priority queue after adding the first potential quadratization sets

Then, the algorithm dequeues the set $\{u_x^2\}$ and tries to quadratize the system with it. The re-written form of Eq. (3.7) using this set is the following:

$$u_t = w_0^2, \tag{3.8}$$
$$w_{0_t} = 2u_x(4u_x^3 u_{xx}) = 4u_x^4 u_{xx}. \tag{3.9}$$

When it verifies that $\{u_x^2\}$ is not a quadratization ($\deg(w_{0t}) > 2$), then it enqueues the subproblems of the node $\{u_x^2\}$ as shown in Figure 3.8, in essence, the sets of variables that could quadratize Eq. (3.9). Then, the algorithm dequeues the set $\{u_x^3\}$. With this set, it finds a quadratization for the system, so it finishes the search and returns $\{u_x^3\}$ as an optimal quadratization.

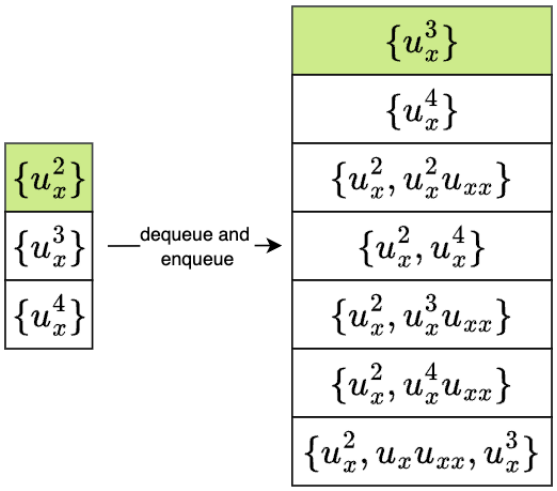

Figure 3.8: State of the priority queue after first dequeue

3.2 Verification of a Quadratization

To design the module for verifying if a set of new variables is a quadratization up to an order p of derivatives, we introduce three sets: V, V^2, and NS, defining them similarly as in the work of Bychkov and Pogudin [10].

Definition 3.2 *(Definition of the sets V, V^2 and NS)*

Consider

$$w_0\left(u, \frac{\partial u}{\partial x}, ..., \frac{\partial^n u}{\partial x^n}\right), \ldots, w_m\left(u, \frac{\partial u}{\partial x}, ..., \frac{\partial^n u}{\partial x^n}\right)$$

as new variables that are quadratization for a given PDE with one unknown function (for simplicity of notation). Here, n is the higher order of the derivatives, and m is the number of variables introduced.

- *V is the set of variables that are a potential quadratization, consisting of the polynomials*

$$1, u, \frac{\partial u}{\partial x}, \ldots, \frac{\partial^{n+p} u}{\partial x^{n+p}}, w_0, \ldots, w_m, \frac{\partial w_0}{\partial x}, \ldots, \frac{\partial^p w_0}{\partial x^p}, \ldots, \frac{\partial w_m}{\partial x}, \ldots, \frac{\partial^p w_m}{\partial x^p}$$

  with p the number of differentiations calculated in the independent variable x.

- *V^2 is the set defined by $\{v_1 v_2 \mid v_1, v_2 \in V\}$.*

- *NS is the set of polynomials from the PDE right-hand side expressions that could not be expressed in a quadratic form using linear combinations of V^2 elements.*

Considering the introduced sets V, V^2, and NS, the general idea is: if the set $\{w_0, ..., w_l\}$ is a quadratization for a given PDE, then each equation within the system can be expressed as a linear combination of elements in V^2. Therefore, the problem is reduced to verifying if each right-hand side expression within the equation system is a linear combination of V^2.

Moreover, let us first define a base that follows an ordering of all the monomials within V^2. This base takes the form:

$$\begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_q \end{bmatrix} \tag{3.10}$$

where $m_1, m_2, \ldots, m_q$ are all unique monomials present in the set V^2, and the restriction $m_1 < m_2 < \cdots < m_q$ holds according to the monomial ordering defined. Now, let us express polynomials as vectors where each non-empty row represents a term present in the specific polynomial as follows

$$\mathbf{v} = \begin{cases} v_i = term_i & \text{if monomial } m_i \text{ is within the polynomial} \\ v_i = 0 & \text{otherwise,} \end{cases} \tag{3.11}$$

and assume that in the root of the search tree (Figure 3.2), the elements in NS are the non-quadratic right-hand side expressions of the original system. Then, the goal is to verify if for every $i \in \{1, \ldots |NS|\}$ the equality $NS_i + \sum_{k=0}^{n} \alpha_k \cdot c_k = 0$ is true, where $c_k \in V^2$, $n = |V^2|$ and $\{\alpha_i, \ldots \alpha_n\}$ is a set of scalars.

Then, to design the algorithm that verifies this proposition, let us define a matrix composed of all the polynomials in V^2 in its vector representation, as in Eq. (3.12). In this matrix, each polynomial vector is expressed using the base defined in Eq. (3.10) and is transposed to constitute the rows of the matrix. In Figure 3.9 we show an example of this representation for a given V^2 of size four.

$$\mathbf{M} = \begin{cases} a_{i,j} = term_j & \text{if monomial } m_j \text{ is present in polynomial } i \\ a_{i,j} = 0 & \text{otherwise} \end{cases} \tag{3.12}$$

$$\begin{array}{c} \\ pol_1 \\ pol_2 \\ pol_3 \\ pol_4 \end{array} \begin{bmatrix} term_1 & 0 & 0 & 0 \\ 0 & 0 & term_3 & 0 \\ 0 & term_2 & 0 & 0 \\ 0 & 0 & 0 & term_4 \end{bmatrix}$$

Figure 3.9: Example of the matrix representation of a given V^2.

Now, the idea of having this matrix representation is to apply the Gaussian elimination method. In this setting, we want to reduce every polynomial vector in NS using rows from the V^2 matrix. To try the reduction only once with each polynomial in V^2 and guarantee that we obtain an irreducible expression for each of them, we will transform the matrix representation of V^2 to a reduced row-echelon form (RREF) [2]. Then, we ensure that V^2 is linearly independent and the resulting polynomials in NS are irreducible.

Given the sets V and V^2, if one of the expressions to quadratize in NS is not a polynomial that can be simply expressed using just one element from V^2, the designed algorithm does the following:

1. First, modify V^2 by applying the Gaussian elimination method to reduce the polynomials based on their leading terms. Throughout this process, maintain the polynomial representation rather than switching to the matrix representation. In detail, the performed operations are:

   1.1 Iterate over every polynomial p_k with $\{p_1, \dots p_n\} \in V^2$. In each iteration:

   i. Start an iteration over $\{p_1, \dots p_{k-1}\}$. For every p_j in this set, first check if the leading term from p_j is present in p_k. If that is the case, try to reduce p_k by multiplying it with the coefficient c of the leading term of p_j in p_k, then do the subtraction $p_k = c \cdot p_k - p_j$. In consequence, the leading term of p_j will be removed from p_k.

   ii. Remove the leading term coefficient of the new reduced polynomial p_k.

   iii. Finally, perform another loop for every p_j with $j < k$, this time reducing p_j, with respect to p_k.

2. Then, for every polynomial NS_i in NS try the following:

2.1 If NS_i is a polynomial that can be simply expressed using elements within V^2 (without performing any linear combination between them), remove it from NS and keep iterating over the loop. If this is not the case:

i. With the reduced set V^2, try to reduce the problematic expressions similarly as explained above. Iterate over the reduced set V^2, and for each polynomial, try to reduce NS_i with the same algorithm used for reducing V^2.

ii. If the expression obtained from the reduction for NS_i is equal to zero, then remove this polynomial from NS. Otherwise, save this reduced form in NS.

3. After the loop ends, check the size of NS. If the set is empty, the variables $\{w_0, ..., w_m\}$ are a quadratization of the system, and the new representation of the PDE is returned. Otherwise, the set of proposed variables is not a quadratization; therefore, return NS, the set of reduced polynomials that could not be transformed into quadratic form.

To understand the algorithm in more detail, we offer the following example:

Example Consider the same PDE introduced in example Eq. (3.1). We introduce the new variable $w = u^2$ and calculate its derivative in x: $w_x = 2u_x u$. Also, we write the new differential equation $w_t = 2u(u_x^2 u) = 2u^2 u_x^2$. In this case, $V = \{1, u, u_x, u^2, 2u_x u\}$ and therefore

$$V^2 = \{1, u, u_x, u^2, 2u_x u, u_x u, u^3, 2u_x u^2, u_x^2, u^2 u_x, 2u_x^2 u, u^4, 2u_x u^3, 4u_x^2 u^2\}.$$

. Thus, if we define our monomial base as:

$$\begin{bmatrix} 1 \\ u \\ u_x \\ u^2 \\ u_x^2 \\ u_x u \\ u^3 \\ u^2 u_x \\ u_x^2 u \\ u^4 \\ u_x u^3 \\ u_x^2 u^2 \end{bmatrix}, \tag{3.13}$$

the matrix representation of V^2 is:

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & u & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & u_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & u^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 2u_xu & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & u_xu & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & u^3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2u_xu^2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & u_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & u^2u_x & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2u_x^2u & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u^4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2u_xu^3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4u_x^2u^2
\end{bmatrix}. \tag{3.14}$$

We notice that neither u_t nor w_t can be easily expressed using original elements from V^2, indicating that it is necessary to perform linear combinations between the polynomials in V^2 to reduce u_t and w_t. Then, we perform the reduction algorithm in V^2 and we obtain the set:

$$\overline{V^2} = \{1, u, u_x, u^2, 0, u_xu, u^3, 0, u_x^2, u^2u_x, u_x^2u, u^4, u_xu^3, u_x^2u^2\},$$

with the corresponding RREF matrix representation:

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & u & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & u_x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & u^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & u_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & u_xu & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & u^3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & u^2u_x & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_x^2u & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u^4 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_xu^3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & u_x^2u^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}. \tag{3.15}$$

Finally the reduction of our problematic expressions (u_t, w_t) is as follows:

$$\overline{u_t} = u_x^2u - \overline{V^2}_{11} = u_x^2u - u_x^2u = 0, \tag{3.16}$$

$$\overline{w_t} = 2u_x^2u^2 - 2\overline{V^2}_{14} = 2u_x^2u^2 - 2 \cdot u_x^2u^2 = 0. \tag{3.17}$$

As $\overline{u_t}$ and $\overline{w_t}$ are both equal to zero and therefore $NS = \emptyset$, we say that w is a quadratization for the PDE in Eq. (3.1).

3.2.1 Other Methods

Another method was also implemented to do this verification. This one is based on constructing a system of linear equations and then solving it through its matrix form with the Gauss-Jordan elimination method. A base following a monomial order is built using all the monomials in the V^2 set. This base is of the same form as shown in Eq. (3.10). Then, the system of equations is

$$A\lambda = b. \tag{3.18}$$

In Eq. (3.18), $A \in \mathbb{R}^{m \times n}$ is a matrix where each column is a polynomial in V^2 expressed with respect to the monomial base where each entry is the monomial coefficient. Further, $\lambda \in \mathbb{R}^n$ is a vector of the coefficients that satisfy the system of equations, in this case, the unknown variables. Finally, $b \in \mathbb{R}^m$ is a vector that represents each expression in NS in terms of the base (Eq. (3.11)), with each entry being the coefficient of that specific monomial. Therefore, if the expression can be written using elements in V^2, then a λ exists that satisfies the system of equations.

3.2.2 Implementations

Three different methods were implemented to do this verification. The first one used the matrices A, λ, and b described in Eq (3.18) to build a system of equations. To solve this system, we used the function `linsolve` from the *Sympy* library [35], which internally implements an algorithm that considers the sparsity of matrix A. This is our case because not every monomial within the system is present in each polynomial of V^2. With this algorithm, the solver's complexity depends only on non-zero entries [35] instead of the generic Gauss-Jordan complexity ($O(n^3)$). However, to compute the input system, we still need to create and store the sparse matrix and vectors A, b, and λ, which grow with the number of monomials in the system, making this solution timely complex.

The second method implemented is the polynomial reduction algorithm described in Section 3.2, using the dense polynomial representation of the *SymPy* library [35], which treats polynomials as Python lists. Despite being more efficient in terms of time and memory than the method described in Section 3.2.1, it still takes a considerable amount of time when dealing with a larger number of variables.

The third method also implemented the polynomial reduction via the Gaussian elimination algorithm, but this time, it used the sparse representation of polynomials in the Sympy library, which represents polynomials as dictionaries. Since these dictionaries considerably reduce manipulation time, better results were obtained in terms of efficiency using this implementation. A graph comparing the results obtained in terms of execution time is shown in Section 4.1 (Figure 4.1).

3.3 Rational Functions

We encounter rational functions in the governing equations of multiple nonlinear PDE models. This can become problematic if we want to find a quadratization for any of these models, as this method can only be applied to polynomial equations. To solve this, we could attempt to transform the right-hand side equations into polynomials by introducing new variables to the system as we do to solve the quadratization problem (polynomialization procedure). For example, if we consider the PDE $u_t = \dfrac{1}{u+1}$, we could introduce a new variable $y = \dfrac{1}{u+1}$ and our system would turn into a two equations PDE: $u_t = y$ and $y_t = -\dfrac{1}{(u+1)^3} = -y^3$. We now have a PDE with polynomial equations, so we can try to run our quadratization algorithm.

However, the problem with this approach is that the algorithm ignores the relation $y = \dfrac{1}{u+1}$ as we introduce the PDE modified by hand, which would incur in disregarding some simplifications such as $(u+1)\cdot y = 1$. This problem then affects the capacity of the algorithm to find lower-order quadratizations for the original PDE, as it could identify that a polynomial is not quadratic when it is according to a fraction relation.

To address this type of PDEs and to make sure the algorithm does the simplifications mentioned above, we implemented a module that finds the partial fraction decomposition of a multivariate rational function, introduces new variables to the system according to this decomposition, and retains the definition of these new variables for further calculations related to the quadratization steps. The overview of this algorithm is as follows:

1. First, find the fraction decomposition of the multivariate rational function present in the PDE of the form $r(u_1, \dots) \in K[u_1, \dots]$, with $u_1, \dots$ being the undefined functions of the PDE. To do this, follow the steps described in the work of Heller and von Manteuffel [23, Section 3.1], which translates into:
   1.1 Re-write every left-hand side equation within the PDE as $\dfrac{n}{d}$. If, in an equation p, d is identified as a function of $(u_1, \dots)$, then p has the form $\dfrac{n(u_1, \dots)}{d(u_1, \dots)}$. Then, cancel common factors in n and d and factorize d over K.
   1.2 For each factor d_i with $i \in [1, \dots, m]$ introduce a new variable q_i such that $q_i = \dfrac{1}{d_i}$. This new set $\{q_1, \dots, q_m\}$ represents the new variables needed for polynomializing the PDE.
   1.3 Now, calculate the Gröbner basis [7] of the ideal I generated by the set $\{q_1 d_1 - 1, \dots, q_m d_m - 1\}$.
   1.4 For every equation within the PDE, express the left-hand side in terms of the set $\{q_1, \dots, q_m\}$ and then reduce it with respect to the calculated Gröbner basis.
2. Then, calculate the first order derivative with respect to the first independent variable for every q_i introduced to obtain the new polynomial PDE system.

3. After every basic algebraic operation (multiplication, division, etc) between polynomials coming from the PDE equations, perform a reduction to the results with respect to the Gröbner basis.

By performing step three, we guarantee that every expression coming from differentiations or basic operations that the algorithm handles is irreducible with respect to the fractional relations we are adding for the polynomialization step. To illustrate the algorithm's instructions more clearly, let us look at an example.

Example Consider the following PDE with time (t) and space (x) independent variables:

$$u_t = \frac{u_x}{u}. \tag{3.19}$$

As it is already expressed in the form $\frac{n(u_1,\dots)}{d(u_1,\dots)}$, we immediately introduce the new variable $q = \frac{1}{u}$. Then the set for the ideal I we will use for the Gröbner basis is $\{q-1\}$. Now, we calculate the Gröbner basis using Buchberger's Algorithm [7, Section 6]. After expressing the original PDE in terms of q, we calculate the derivative with respect to t of q and reduce all results with respect to the Gröbner basis. Then, we get the following PDE:

$$\begin{aligned} u_t &= u_x q \\ q_t &= -q^3 u_x. \end{aligned}$$

We perform the quadratization algorithm, obtaining the following rewritten form of our original PDE:

$$\begin{aligned} u_t &= u_x q \\ q_t &= q_x q. \end{aligned}$$

3.4 Observation on the Search Success of the Algorithm

As established in Section 2.2.2, we know that for every PDE system there is a quadratization. However, our approach also depends on the number of partial differentiations we perform on the new variables, as they are also part of the rewritten form of the system coming from the transformation procedure. This could result in the algorithm not finding a quadratization because either the number of partial differentiations of the new variables was not enough or the bound defined for the number of nodes to traverse did not suffice. One approach we could take to guarantee a successful search is the following:

- Fix N to a high number (for example $N = 100$) and fix an arbitrary number of differentiations (for example $p = 5$)
- Run any of the search algorithms with the chosen p until it either finds a result or traverses N nodes. If a quadratization was found, return it.
- Else, set $p = p + 1$ and $N = 2 \cdot N$, and go back to step 2.

Moreover, we can analyze the complexity of this algorithm. Let us suppose that every search for each p and N takes $O(N^c)$ where c is a constant. Then, if N_0 is the minimum value of N we tried, the final cost of the algorithm is

$$\sum_{i=1}^{m} (N_0 \cdot 2^i)^c = O((N_0 \cdot 2^m)^c) = O(N^{*^c}), \tag{3.20}$$

where N^* is the value of N at which we find a quadratization.

Chapter 4

Experiments and Results

To analyze the performance of *QuPed*, we ran some tests over practical PDE examples with different conditions. The results were obtained on a personal laptop with the specifications Apple M2, macOS Sonoma 14, and Python version 3.10.12. It is also worth mentioning that our implementation only supports one-dimensional PDEs (two independent variables). The link to the repository with the implemented software in Python is `https://github.com/albaniolivieri/pde-quad.git`.

4.1 PDE Models Used

We describe some practical examples of PDE models used to test the algorithm:

- Dym equation [32]: The Harry Dym equation is an important dynamical equation that is integrable and finds applications in several physical systems. The Dym equation represents a system in which dispersion and non-linearity are coupled together:

$$u_t = u^3 u_{xxx}. \tag{4.1}$$

- Variation of non-adiabatic tubular reactor model [31]: This model describes species concentration and temperature evolution in a single reaction:

$$u_t = \frac{1}{Pe} u_{ss} - u_s - \mathcal{D} f(v) \tag{4.2}$$

$$v_t = \frac{1}{Pe} v_{ss} - v_s - \beta(v + \theta_{ref}) + \mathcal{B}\mathcal{D} f(v), \tag{4.3}$$

  where $f(v)$ is a polynomial nonlinear term defined as $f(v) = up$ with $p = c_0 + c_1 v + c_2 v^2 + c_3 v^3 + \dots$ a Taylor expansion of an exponential function. In this section, we show results with p of degree three and four. Also, $\mathcal{D}$ is Damköhler number, Pe is Pèclet number and $\mathcal{B}, \beta, \theta_{ref}$ are known constants.

- Modified KdV equation (mKdV) [44]: The KdV equation is a generic model for the study of weakly nonlinear long waves, incorporating leading order nonlinearity and

dispersion. Also, it describes surface waves of long wavelength and small amplitude in shallow water:

$$u_t = au^2u_x - u_{xxx}, \tag{4.4}$$

with $a > 0$.

- Solar wind model [28]: The HUX (Heliospheric Upwinding eXtrapolation) model is a two-dimensional time-stationary model that predicts the heliospheric solar wind speed:

$$u_r = \frac{\Omega_{rot}u_\phi}{u}, \tag{4.5}$$

with Ω_{rot} the angular frequency on the Sun's rotation evaluated at a constant Carrington latitude.

- Schlögl model [8]: Schlögl's model is a simple example of a chemical reaction system that exhibits bistability. Next, we show a version of this model without input functions:

$$u_t = u_{xx} - k(u - u_1)(u - u_2)(u - u_3), \tag{4.6}$$

with $k \geq 0$ and $u_1 < u_2 < u_3$.

- Euler equations [27]: The Euler equations are derived from tile physical principles of conservation of mass, momentum, and energy:

$$\rho_t = -u\rho_x - \rho u_x \tag{4.7}$$

$$u_t = -u_x u - \frac{p_x}{\rho} \tag{4.8}$$

$$p_t = -u_x p - up_x. \tag{4.9}$$

- FitzHugh-Nagamo system [5]: A simplified neuron model of the Hodgkin-Huxley model, which describes activation and deactivation dynamics of a spiking neuron:

$$v_t = \varepsilon v_{xx} + \frac{1}{\varepsilon}v(v - 0.1)(1 - v) - \frac{1}{\varepsilon}u + \frac{1}{\varepsilon}q \tag{4.10}$$

$$u_t = hv - \gamma u + q, \tag{4.11}$$

with $\varepsilon = 0.015$, $h = 0.5$, $\gamma = 2$, $q = 0.05$.

- Schnakenberg equations [34]: Evolution equations for reaction-diffusion systems with cross-diffusion:

$$u_t = D_u u_{xx} + D_{uv}v_{xx} + k_1a_1 - k_2u + k_3u^2v \tag{4.12}$$

$$v_t = D_v v_{xx} + D_{vu}u_{xx} + k_4b_1 - k_3u^2v, \tag{4.13}$$

where $D_u > 0$, $D_v > 0$, D_{uv} and D_{vu} are diffusion and cross-diffusion coefficients respectively, and a_1, b_1, k_1, k_2, k_3, k_4 are all positive constants.

- Brusselator system [33]: The Brusselator system was developed to model morphogenesis and pattern formation in chemical reactions:

$$u_t = d_1u_x + \lambda(1 - (b + 1)u + bu^2v) \tag{4.14}$$

$$v_t = d_2v_x + \lambda a^2(u - u^2v), \tag{4.15}$$

with d_1, d_2, λ, a and b positive constants.

- Allen–Cahn equation [18]: Describes the motion of antiphase boundaries in crystalline solids. It was proposed as a simple model for the process of phase separation of a binary alloy at a fixed temperature:
$$u_t = u_{xx} + u - u^3. \tag{4.16}$$
- 1D Nonlinear Heat equation [21]: Among the most widely studied models, it presents a rich mathematical structure for studying "blow-up" functions:
$$u_t = u_{xx} + u^p, \tag{4.17}$$
with $p > 1$.

4.2 Comparison between Implementations for Verifying a Quadratization

To compare the efficiency between the methods described in Section 3.2.2, we tried to confirm that the variables $w_0 = u^3$ and $w_1 = u_x^2 u$ were a quadratization for the equation $u_t = u^3 u_{xxx}$ (Dym equation [32]). We increased the differentiation order of the new variables introduced to see how each algorithm performed in terms of time with the growth of V^2 cardinality. The results of this experiment are shown through the graph in Figure 4.1 (to see explicit results, see Table A.1 in the Appendix).

In Figure 4.1, the x-axis represents the number of differentiations calculated for w_0 and w_1, and the y-axis is the time that it took to deliver a response for each of the cases. From this graph, it is clear that the Gauss-Jordan method with the matrix representation is the one that performed worst, growing exponentially with the number of monomials in V^2. The wide difference between this implementation and the other two could result from ignoring the sparsity of the matrix representation of V^2 (as explained in Section 3.2.1).

4.3 General Results for Branch & Bound and Nearest Neighbor Algorithms

In Table 4.1, we show the results obtained by running *QuPed* for the examples described in Section 4.1, using both search algorithms B&B and Nearest Neighbor (NN), and the best sorting heuristic for each example.

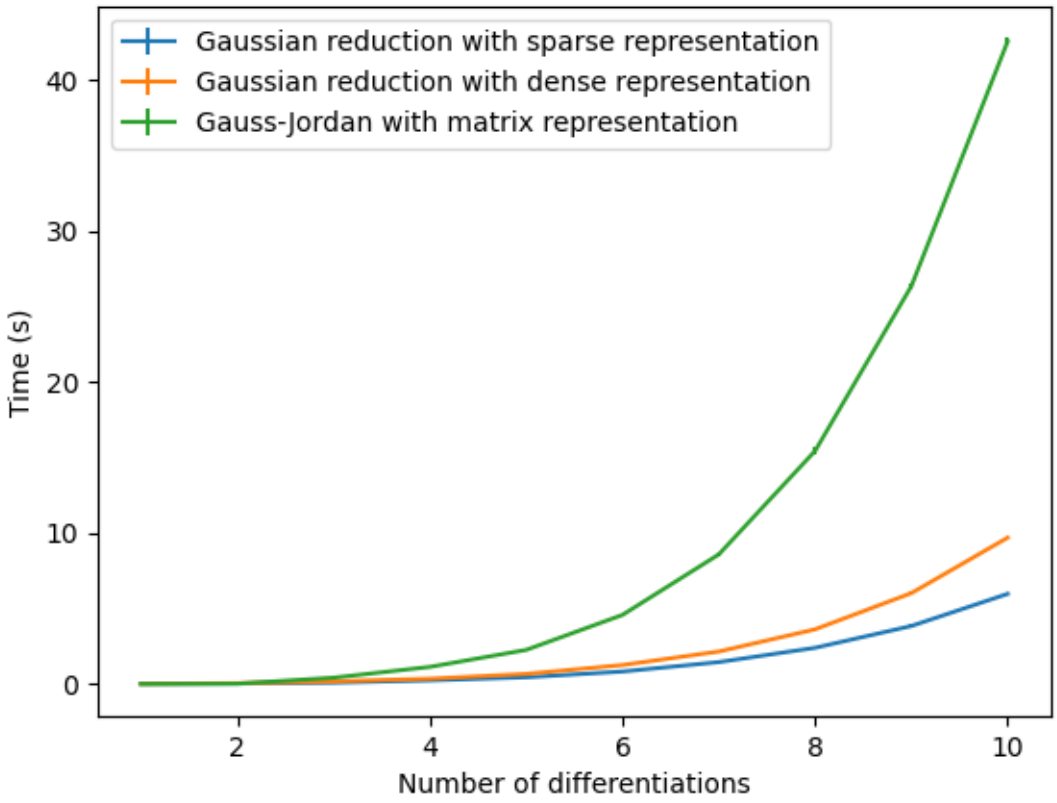

Figure 4.1: Efficiency comparison between quadratization verification algorithms.

Table 4.1: New variables introduced, time elapsed using the best heuristic, and number of nodes traversed for each example using B&B search.

PDE system	New variables B&B	Time B&B (s)	Nodes traversed B&B	New variables NN	Time NN (s)	Nodes traversed NN
Dym equation	u^3, $u_x^2 u$	0.64 ± 0.04	37	u^3, $u_x^2 u$	1.16 ± 0.01	10
Tubular reactor	uv, v^2, v^2u, v^3	1874.77	12793	uv, v^2, v^2u, v^3	2284.24	10158
mKdV equation	u^2	0.04 ± 0.03	4	u^2	0.03 ± 0.004	2
Solar wind model	$1/u$	0.01 ± 0.0001	1	$1/u$	0.02 ± 0.0002	1
Schlögl model	u^2	0.05 ± 0.03	3	u^2	0.04 ± 0.0003	2
Euler equations	$1/\rho$	0.11 ± 0.03	1	$1/\rho$	0.10 ± 0.001	1
FitzHugh-Nagamo system	v^2	0.17 ± 0.04	3	v^2	0.16 ± 0.001	2
Schnakenberg equations	uv, u^2	1.17 ± 0.04	11	uv, u^2	1.17 ± 0.02	8
Brusselator system	u^2, uv	0.82 ± 0.03	11	u^2, uv	0.81 ± 0.005	8
Allen-Cahn equation	u^2	0.04 ± 0.03	3	u^2	0.04 ± 0.0002	2
Heat equation ($p = 6$)	u^3, $u_x u$, u^5	3.02 ± 0.13	50	u^3, u_x^2, u^5	1.66 ± 0.004	17

In most of the examples, the performance in terms of efficiency is more or less the same for both algorithms. We see greater differences in the Heat equation (Eq. (4.17)), the Dym equation (Eq. (4.1)), and the Tubular reactor model (Eq. (4.2)). Some of these differences are due to the implementation of the shrink function (Section 3.1.1). With this functionality, the B&B algorithm can find a high-order quadratization and, from the subsets of this suboptimal solution, identify a better quadratization, helping to quickly prune many branches (including those of the same order as the optimal one).

Particularly, in the Dym equation execution, the B&B algorithm performed best, where it skipped almost every node with potential quadratizations of order two or higher by finding an optimal one through the shrink function in the first branch traversed. Meanwhile, the Nearest Neighbor algorithm visited multiple nodes with variable sets of cardinality two before arriving at an optimal quadratization. On the contrary, the Nearest Neighbor algorithm yielded the best result for the Heat equation example. In this case, the B&B algorithm found a suboptimal quadratization before reaching the best one, which forced it to run the shrink function twice. Alternatively, in the Nearest Neighbor execution, the algorithm only

had to run this function once.

On the other hand, in the Tubular reactor example, the difference in times can be explained by the cost of the insertion operation in large priority queues, which is the main data structure used in the Nearest Neighbor approach. Given that the best quadratization for this system is of order four, the treewidth grew considerably before arriving at the optimal solution, implying the number of nodes that were inserted into the priority queue was significantly high.

4.4 Effect of Shrink Function

To highlight the impact on the efficiency of using the shrink function within the B&B algorithm and the Neatest Neighbor approach, we ran executions with some examples with and without the use of this function. The results of these experiments are shown in Table 4.2 for the B&B algorithm and Table 4.3 for the Nearest Neighbor approach.

Table 4.2: Performance of B&B search with and without the use of the shrink function.

PDE system	with shrink function (s)	without shrink function (s)
Dym equation	0.62 ± 0.04	0.75 ± 0.03
Schnakenberg equations	1.19 ± 0.04	0.99 ± 0.04
Brusselator system	0.82 ± 0.04	0.69 ± 0.03
Heat equation ($p = 6$)	3.07 ± 0.04	2.50 ± 0.04

Table 4.3: Performance of Nearest Neighbor search with and without the use of the shrink function.

PDE system	with shrink function (s)	without shrink function (s)
Dym equation	1.12 ± 0.01	9.32 ± 0.06
Schnakenberg equations	1.15 ± 0.01	0.97 ± 0.02
Brusselator system	0.81 ± 0.01	0.66 ± 0.01
Heat equation ($p = 6$)	2.20 ± 0.02	1.93 ± 0.02

When analyzing results in Table 4.2, we can see the impact of the shrink function in the B&B search approach. While in the Schnakenberg (Eq. (4.12)), Brusselator (Eq. (4.14)), and Heat equation (Eq. (4.17)) examples, the algorithm shows a better performance without the shrink function, meaning that it does not need it to find an optimal quadratization quicker, the difference in time is in the order of milliseconds. On the other hand, if we compare the execution times for the Dym equation (Eq. (4.1)), we can see that the implementation of the shrink function had a high positive impact.

In Table 4.3, we see more or less the same result pattern for the Nearest Neighbor algorithm. Furthermore, the distance between both executions for the Dym example is broader than in the B&B search, and more importantly, as the optimal quadratization of order two

comes from the subsets generated by the shrink equation and not the general decompositions, the best quadratization it finds without the shrink function is of order three for both algorithms, which highlights the importance of this functionality.

Thus, it is shown that implementing this step can yield better results in terms of better quadratizations for the B&B and Nearest Neighbor search algorithms in general, making it worthwhile to sacrifice small efficiency differences in cases where the algorithms do not need the shrink function to find an optimal quadratization.

4.5 Sorting Heuristics Comparison

We compared the three implemented sorting functions to analyze how sensible both algorithms (B&B and Nearest Neighbor) are in terms of efficiency to the sorting heuristics (Section 3.1). The B&B and Nearest Neighbor algorithm results are shown in Table 4.4 and Table 4.5, respectively. In each table, h_1 is the heuristic *by order and degree*, h_2 is *by degree and order* and h_3 is the heuristic given by the function $degree + 2 \cdot order$.

Table 4.4: Comparison of sorting heuristics used in B&B algorithm.

PDE system	h_1 (s)	h_2 (s)	h_3 (s)
Dym equation	0.65 ± 0.06	0.63 ± 0.03	0.61 ± 0.01
mKdV equation	0.04 ± 0.03	0.03 ± 0.001	0.03 ± 0.002
Solar wind	0.02 ± 0.03	0.01 ± 0.0002	0.01 ± 0.001
Schlögl model	0.05 ± 0.03	0.04 ± 0.0002	0.04 ± 0.003
Euler equations	0.11 ± 0.03	0.10 ± 0.002	0.10 ± 0.01
FitzHugh-Nagamo system	0.17 ± 0.04	0.16 ± 0.01	0.15 ± 0.002
Schnakenberg equations	1.18 ± 0.03	1.17 ± 0.03	1.18 ± 0.03
Brusselator system	0.85 ± 0.05	0.81 ± 0.01	0.81 ± 0.01
Allen-Cahn equation	0.04 ± 0.03	0.04 ± 0.002	0.04 ± 0.001
Heat equation ($p = 6$)	3.16 ± 0.04	3.02 ± 0.05	3.23 ± 0.14

Table 4.5: Comparison of sorting heuristics used in Nearest Neighbor algorithm.

PDE system	h_1 (s)	h_2 (s)	h_3 (s)
Dym equation	1.12 ± 0.01	1.13 ± 0.01	1.13 ± 0.004
mKdV equation	0.03 ± 0.0002	0.03 ± 0.0002	0.03 ± 0.0002
Solar wind	0.01 ± 0.0001	0.01 ± 0.0001	0.02 ± 0.0003
Schlögl model	0.04 ± 0.001	0.04 ± 0.0002	0.04 ± 0.001
Euler equations	0.09 ± 0.003	0.10 ± 0.01	0.10 ± 0.003
FitzHugh-Nagamo system	0.15 ± 0.004	0.15 ± 0.001	0.15 ± 0.001
Schnakenberg equations	1.16 ± 0.03	1.15 ± 0.01	1.15 ± 0.01
Brusselator system	0.81 ± 0.01	0.81 ± 0.01	0.81 ± 0.01
Allen-Cahn equation	0.04 ± 0.0002	0.04 ± 0.0002	0.04 ± 0.0004
Heat equation ($p = 6$)	2.23 ± 0.05	2.20 ± 0.02	2.23 ± 0.05

The results of this experiment in both cases suggest that neither of the search algorithms is highly sensitive to the selected sorting heuristic. The difference between execution times for each example is in the order of milliseconds in both B&B and Nearest Neighbor approaches. However, we do notice that executions with h_2 are the fastest in almost every instance, which corresponds to the combinatorial result described in Section 3.1, where we show that a higher polynomial degree translates into more branches (wider treewidth) in the search tree.

4.6 Effect of Pruning Rules

As described in Section 3.1.1, we implemented two pruning rules for the B&B search approach. The first one is given by the smallest order quadratization found, essential for bounding and pruning branches. The second pruning rule is related to the order of the derivatives allowed within the new variables. Table 4.6 shows time comparisons with and without implementing the second pruning rule for a group of PDE examples.

Table 4.6: Performance of B&B search with and without the derivative-order pruning rule.

PDE system	with prune rule (s)	without prune rule (s)
Dym equation	0.64 ± 0.03	1.28 ± 0.03
mKdV equation	0.03 ± 0.001	0.04 ± 0.03
Solar wind	0.05 ± 0.002	0.06 ± 0.03
Schlögl model	0.04 ± 0.0003	0.04 ± 0.03
Euler equations	0.09 ± 0.0002	0.10 ± 0.03
FitzHugh-Nagamo system	0.15 ± 0.001	0.16 ± 0.04
Schnakenberg equations	1.20 ± 0.04	1.20 ± 0.06
Brusselator system	0.80 ± 0.01	0.84 ± 0.05
Allen-Cahn equation	0.04 ± 0.002	0.04 ± 0.03
Heat equation ($p = 6$)	3.02 ± 0.03	3.06 ± 0.03

Looking at Table 4.6, we can see that for all the examples, the algorithm does better or the same with the derivative-order prune rule implemented. Particularly for the Dym equation, the implementation of this prune rule yields a significant speed-up.

4.7 Comparison with Other Approaches

4.7.1 Comparison with *QBee* Dimension-agnostic Quadratization

In the next table, we compare results obtained from the *QBee* algorithm used for semi-discretized PDEs [9, Algorithm 5.3 Section 5] with our algorithm's performance on the solar wind, tubular reactor, and Allen-Cahn equations, in which a discretization was done in the works of Issan and Kramer [28], Kramer and Willcox [30], and Yang et al. [45], respectively. For this experiment, we executed both algorithms on the same machine.

Table 4.7: Comparison between *QBee* dimension-agnostic quadratization and *QuPed*

PDE system	*QBee*		*QuPed*	
	Time (s)	Quad order	Time (s)	Quad order
Solar wind model	0.34	4	0.06	1
Allen-Cahn equation	0.07	1	0.04	1
Tubular reactor model	0.68	4	1874.77	4

Looking at Table 4.7, we can see significant differences in the results of both software. In the solar wind example, our algorithm outperforms *QBee* in both time and number of variables; additionally, we get a PDE quadratic system of dimension two (two differential equations), in contrast with the system resulting from using *QBee*, where the final quadratic PDE is of dimension six [9]. For the Allen-Cahn equation, we also do better in terms of time. On the other hand, for the tubular example case, our algorithm outputs the same number of variables but does considerably worse in terms of time efficiency.

It is also important to highlight that we can only compare both algorithms with a few examples, as the discretization procedure for a PDE is not trivial; it depends on the PDE's structure and is a field of study by itself.

4.7.2 Comparison with Quadratizations done by hand

As discussed earlier, the quadratization procedure for PDEs is useful for solving certain problems and is often used as a preprocessing step. In the majority of these works, this procedure was done by hand. In Table 4.8, we compare the quadratizations shown in the works of Kramer and Willcox ([30, Section 4], [31, Section 3.2]), and Qian et al. [39, Section 5.2] with results obtained from our algorithm.

Table 4.8: Comparison of lifting procedures done by hand and the quadratizations found by our algorithm

PDE system	Lifting by hand	Lifting with *QuPed*
Euler equations	$1/\rho$	$1/\rho$
FitzHugh-Nagamo system	v^2	v^2
Tubular reactor model	uv, v^2u, v^3u, v^2, v^3	uv, v^2, v^2u, v^3

While analyzing Table 4.8, we notice that our algorithm found a better quadratization for the tubular reactor model, yielding a quadratization of order four instead of five. For the other examples, our algorithm found the same quadratizations, and for both the Euler equations and FitzHugh-Nagamo system, it outputs the solution in less than half a second.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

To solve various engineering and science problems, the concept of variable transformation has been widely explored to bring a complex problem to a simpler plane, where analysis, manipulation, and study become easier tasks. While this technique can be applied to problems modeled with ODEs using several available tools, a wide range of more complex dynamics must be modeled with PDEs to capture all their characteristics. However, limited tools are available to handle these types of problems in the context of variable transformation or lifting.

In this regard, we present the software *QuPed*, an algorithm for finding optimal quadratizations for PDE systems. Throughout this work, we described its construction, where we relied on implementing optimization algorithms such as the branch-and-bound framework and nearest-neighbor search. Moreover, we included additional tools such as the polynomialization step for rational functions, which enabled us to handle a wider range of scientific nonlinear models.

Additionally, we measured the performance of *QuPed* by conducting a benchmark study, where we tested the algorithm and its variations (branch-and-bound and nearest neighbor search) with several PDE models from practical applications and scientific studies found in the literature. Throughout these experiments and tests, we obtained quadratic transformations for complex examples that were out of reach using state-of-the-art tools, such as the Dym equation (Eq. (4.1)), Heat equation (Eq. (4.17)), and the Schlögl model (Eq. (4.6)). Furthermore, for most examples, the algorithm (with both the B&B and Nearest Neighbor approaches) delivered an optimal quadratization within the first second of execution. We also depicted the effect of some functionalities added to improve efficiency and the impact of pruning rules in the branch-and-bound case.

Moreover, we also showed the results of alternative implementations built for our algorithm, where we demonstrated that the Gaussian elimination algorithm with a polynomial sparse representation performs best in terms of efficiency for verifying if a set of variables is a quadratization.

Furthermore, when we compared our software performance with the *QBee* dimension-agnostic quadratization, we unveiled considerable differences between the results obtained from both algorithms. We conclude that our algorithm can find quadratizations of fewer auxiliary variables, while the *QBee* approach does not guarantee optimality. Nonetheless, given the unfavorable result in terms of time when handling the Tubular reactor model example, we also confirm that there is room for improvement in terms of efficiency for the algorithm that we proposed in this work.

5.2 Future Work

Our study focuses on the quadratization of polynomial PDEs. However, many scientific problems are models with nonlinear and nonpolynomial PDEs. To address these problems, a polynomialization algorithm must be developed, where the goal is to find a polynomial representation of any system by adding new variables.

Moving forward, we plan to keep searching and studying ways of improving the algorithm's efficiency. In this context, our efforts will be divided into several ideas. We will keep exploring the definition of new pruning rules for the branch-and-bound approach and make changes regarding memory usage for both search algorithms.

Additionally, we aim to develop methods to reduce the number of equations of the resulting quadratic PDE systems. We plan to add a priority system for the monomials used within the final quadratic equations to the reduction step in the verification algorithm. This would pose a considerable improvement, as one of the challenges or downsides of this procedure is the increase of the original system dimension.

Finally, we hope our work will inspire further research into the variable transformation field for PDEs. With all the complex phenomena surrounding us, these tools can potentially increase our general understanding, insight, and analysis of problems that presently are out of reach.

Bibliography

[1] F. Alauddin. Quadratization of ODEs: Monomial vs. non-monomial, 2020.

[2] S. Andrilli and D. Hecker. *Elementary Linear Algebra*. Elsevier Science, 2016.

[3] M. Balajewicz, I. Tezaur, and E. Dowell. Minimal subspace rotation on the stiefel manifold for stabilization and enhancement of projection-based reduced order models for the compressible navier–stokes equations. *Journal of Computational Physics*, 321:224–241, 2016.

[4] P. Benner and T. Breiten. Two-sided projection methods for nonlinear model order reduction. *SIAM Journal on Scientific Computing*, 37:B239–B260, 03 2015.

[5] P. Benner and P. Goyal. Balanced truncation model order reduction for quadratic-bilinear control systems, 2017.

[6] O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.

[7] B. Buchberger. Gröbner bases: A short introduction for systems theorists. In R. Moreno-Díaz, B. Buchberger, and J. Luis Freire, editors, *Computer Aided Systems Theory — EUROCAST 2001*, pages 1–19, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[8] R. Buchholz, H. Engel, E. Kammann, and F. Tröltzsch. On the optimal control of the Schlögl-model. *Computational Optimization and Applications*, 56(1):153–185, 2013.

[9] A. Bychkov, O. Issan, G. Pogudin, and B. Kramer. Exact and optimal quadratization of nonlinear finite-dimensional nonautonomous dynamical systems. *SIAM Journal on Applied Dynamical Systems*, 23(1):982–1016, 2024.

[10] A. Bychkov and G. Pogudin. Optimal monomial quadratization for ODE systems. In P. Flocchini and L. Moura, editors, *Combinatorial Algorithms*, pages 122–136, Cham, 2021. Springer International Publishing.

[11] V. Chepyzhov and M. Vishik. *Attractors for Equations of Mathematical Physics*. American Mathematical Society Colloquium publications. American Mathematical Society, 2002.

[12] M. M. Costa and M. F. Silva. A survey on path planning algorithms for mobile robots. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–7, 2019.

[13] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer New York, 2008.

[14] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32(3):505–536, jul 1985.

[15] T. Dekker and W. Hoffmann. Rehabilitation of the Gauss-Jordan algorithm. *Numerische Mathematik*, 54(5):591–599, 1989.

[16] L. Evans. *Partial Differential Equations*. Graduate Studies in Mathematics. American Mathematical Society, 2022.

[17] F. Fages, G. Le Guludec, O. Bournez, and A. Pouly. Strong Turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In J. Feret and H. Koeppl, editors, *Computational Methods in Systems Biology*, pages 108–127, Cham, 2017. Springer International Publishing.

[18] Y. Fukao, Y. Morita, and H. Ninomiya. Some entire solutions of the Allen–Cahn equation. *Taiwanese Journal of Mathematics*, 8(1):15 – 32, 2004.

[19] G. H. Golub and C. F. Van Loan. *Matrix Computations - 4th Edition*. Johns Hopkins University Press, Philadelphia, PA, 2013.

[20] C. Gu. QLMOR: A projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(9):1307–1320, 2011.

[21] C. Gui, W.-M. Ni, and X. Wang. Further study on a nonlinear heat equation. *Journal of Differential Equations*, 169(2):588–613, 2001.

[22] L. Guillot, B. Cochelin, and C. Vergez. A generic and efficient taylor series based continuation method using a quadratic recast of smooth nonlinear systems. *International Journal for Numerical Methods in Engineering*, 119, 02 2019.

[23] M. Heller and A. von Manteuffel. MultivariateApart: Generalized partial fractions. *CoRR*, abs/2101.08283, 2021.

[24] M. Hemery, F. Fages, and S. Soliman. On the complexity of quadratization for polynomial differential equations, 2020.

[25] N. J. Higham. Gaussian elimination. *WIREs Computational Statistics*, 3(3):230–238, 2011.

[26] G. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. 02 2002.

[27] H. T. Huynh. Accurate upwind methods for the Euler equations. *SIAM Journal on Numerical Analysis*, 32(5):1565–1619, 1995.

[28] O. Issan and B. Kramer. Predicting solar wind streams from the inner-heliosphere to earth via shifted operator inference. *Journal of Computational Physics*, 473:111689, Jan. 2023.

[29] D. C. Kozen. *Depth-First and Breadth-First Search*, pages 19–24. Springer New York, New York, NY, 1992.

[30] B. Kramer and K. E. Willcox. Nonlinear model order reduction via lifting transformations and proper orthogonal decomposition. *AIAA Journal*, 57(6):2297–2307, jun 2019.

[31] B. Kramer and K. E. Willcox. Balanced truncation model reduction for lifted nonlinear systems, 2020.

[32] S. Kumar, M. P. Tripathi, and O. P. Singh. A fractional model of Harry Dym equation and its approximate solution. *Ain Shams Engineering Journal*, 4(1):111–115, 2013.

[33] B. Li and M.-x. Wang. Diffusion-driven instability and hopf bifurcation in brusselator system. *Applied Mathematics and Mechanics*, 29(6):825–832, 2008.

[34] A. Madzvamuse, H. S. Ndakwo, and R. Barreira. Cross-diffusion-driven instability for reaction-diffusion systems: Analysis and simulations. *Journal of Mathematical Biology*, 70(4):709–743, 2015.

[35] A. Meurer, C. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz. SymPy: Symbolic computing in Python. *PeerJ Computer Science*, 3:e103, Jan. 2017.

[36] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[37] E. D. Nardo. Symbolic calculus in mathematical statistics: A review, 2015.

[38] G. Pogudin. Existence of a monomial quadratization for every polynomial pde. unpublished, 2024.

[39] E. Qian, B. Kramer, B. Peherstorfer, and K. Willcox. Lift & Learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406:132401, 2020.

[40] W. A. Strauss. *Partial Differential Equations: An Introduction*. John Wiley & Sons, Brown University, 2007.

[41] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[42] R. Temam. *Navier-Stokes Equations: Theory and Numerical Analysis*. AMS/Chelsea publication. AMS Chelsea Pub., 2001.

[43] L. N. Trefethen and D. Bau. *Numerical Linear Algebra, Twenty-fifth Anniversary Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2022.

[44] A.-M. W. Chapter 9 the KdV equation. In C. M. Dafermos and M. Pokorny, editors, *Handbook of Differential Equations: Evolutionary Equations*, volume 4, pages 485–568. North-Holland, 2008.

[45] J. Yang, Q. Du, and W. Zhang. Uniform Lp-bound of the Allen–Cahn equation and its numerical discretization. *International Journal of Numerical Analysis and Modeling*, 15:213–227, 01 2018.

Appendix A

Results Tables

In this section, we present the result table for the experiments described in Section 4.

Number of differentiations	Matrix representation (s)	Dense polynomial representation (s)	Sparse polynomial representation (s)
1	0.015 ± 0.006	0.034 ± 0.009	0.027 ± 0.025
2	0.034 ± 0.032	0.063 ± 0.007	0.046 ± 0.002
3	0.429 ± 0.041	0.186 ± 0.027	0.104 ± 0.011
4	1.146 ± 0.047	0.362 ± 0.021	0.253 ± 0.003
5	2.271 ± 0.043	0.680 ± 0.023	0.457 ± 0.004
6	4.591 ± 0.054	1.269 ± 0.039	0.839 ± 0.010
7	8.599 ± 0.119	2.162 ± 0.038	1.463 ± 0.015
8	15.455 ± 0.237	3.635 ± 0.062	2.410 ± 0.043
9	26.377 ± 0.207	6.034 ± 0.167	3.856 ± 0.137
10	42.593 ± 0.219	9.711 ± 0.083	5.985 ± 0.084

Table A.1: Comparison between implementation for verifying a quadratization.

Appendix B

Example systems after quadratization

B.1 Dym Equation

$$
\begin{aligned}
&w_0 = u^3 \qquad w_1 = u_x^2 u \\
&u_t = u_{xxx} w_0 \\
&w_{0_t} = w_0 w_{0_{xxx}} - 2 w_{0_x} w_{0_{xx}} + 10 w_{0_x} w_1 \\
&w_{1_t} = w_0 w_{1_{xxx}} - \frac{2}{3} w_{0_{xx}} w_{0_{xxx}} + 4 w_{0_{xx}} w_{1_x} + 4 w_{0_{xxx}} w_1 - 24 w_1 w_{1_x}
\end{aligned}
$$

B.2 Non-adiabatic Tubular Reactor Model

$$
\begin{aligned}
&w_0 = uv \qquad w_1 = v^2 \qquad w_2 = v^2 u \qquad w_3 = v^3 \\
&u_t = -\mathcal{D}Pec_0 u - \mathcal{D}Pec_1 uv - \mathcal{D}Pec_2 w_2 - \mathcal{D}Pec_3 w_0 w_1 - Pe u_s + u_{ss} \\
&v_t = \mathcal{B}\mathcal{D}Pec_0 u + \mathcal{B}\mathcal{D}Pec_1 uv + \mathcal{B}\mathcal{D}c_2 w_2 + \mathcal{B}\mathcal{D}Pec_3 w_0 w_1 - Pe\beta v + Pe\beta\theta_{ref} - Pe v_s + v_{ss} \\
&w_{0_t} = \mathcal{B}\mathcal{D}Pec_0 u^2 + \mathcal{B}\mathcal{D}Pec_1 w_0 u + \mathcal{B}\mathcal{D}Pec_2 w_0^2 + \mathcal{B}\mathcal{D}Pec_3 w_0 w_2 - \mathcal{D}Pec_1 w_2 - \mathcal{D}Pec_2 w_0 w_1 - \\
&DPec_3 w_0 w_3 + Pe\beta\theta_{ref} u - Pe v_s u - Pe u_s v + uv(-\mathcal{D}Pec_0 - Pe\beta) + v_{ss} u + u_{ss} v \\
&w_{1_t} = 2\mathcal{B}\mathcal{D}Pec_0 uv + 2\mathcal{B}\mathcal{D}Pec_1 w_2 + 2\mathcal{B}\mathcal{D}Pec_2 w_0 w_1 + 2\mathcal{B}\mathcal{D}Pec_3 w_0 w_3 - 2Pe\beta v^2 + 2Pe\beta\theta_{ref} v \\
&- Pe w_{1_s} + 2\theta_{ref} v_{ss} \\
&w_{2_t} = 2\mathcal{B}\mathcal{D}Pec_0 u w_0 + 2\mathcal{B}\mathcal{D}Pec_1 w_0^2 + 2\mathcal{B}\mathcal{D}Pec_2 w_0 w_2 + 2\mathcal{B}\mathcal{D}Pec_3 w_2^2 - DPec_1 w_0 w_1 - \\
&\mathcal{D}Pec_2 w_0 w_3 - \mathcal{D}Pec_3 w_2 w_3 + 2Pe\beta\theta_{ref} uv - Pe u_s w_1 - 2Pe v_s w_0 + w_{1_{ss}} u - 4 v_s w_s - \\
&2 v_{ss} w_0 + w_2(-\mathcal{D}Pec_0 - 2Pe\beta) + w_{2_{ss}} \\
&w_{3_t} = 3\mathcal{B}\mathcal{D}Pec_0 w_2 + 3\mathcal{B}\mathcal{D}Pec_1 w_0 w_1 + 3\mathcal{B}\mathcal{D}Pec_2 w_0 w_3 + 3\mathcal{B}\mathcal{D}Pec_3 w_2 w_3 + 3Pe\beta\theta_{ref} v^2 - \\
&3Pe\beta v w_1 - Pe w_{3_s} + 3 w_{1_{ss}} v - w_{3_{ss}}
\end{aligned}
$$

B.3 mKdV Equation

$$
\begin{aligned}
w_0 &= u^2 \\
u_t &= a u_x w_0 - u_{xxx} \\
w_{0_t} &= a w_0 w_{0_x} - 2 u_{xxx} u
\end{aligned}
$$

B.4 Solar Wind Model

$$
\begin{aligned}
w_0 &= \frac{1}{u} \\
u_r &= \Omega_{rot} w_0 u_\phi \\
w_{0_r} &= \Omega_{rot} w_0 w_{0_\phi}
\end{aligned}
$$

B.5 Schlögl Model

$$
\begin{aligned}
w_0 &= u^2 \\
u_t &= k u^2 (u_1 +_2 + u_3) - k w_0 u + k u (-u_1 u_2 - u_1 u_3 - u_2 u_3) + u_{xx} + k u_1 u_2 u_3 \\
w_{0_t} &= 2 k u^2 (-u_1 u_2 - u_1 u_3 - u_2 u_3) + 2 k u_1 u_2 u_3 u + 2 k u w_0 (u_1 + u_2 + u_3) - 2 u_{xx} u - 2 k w_0^2
\end{aligned}
$$

B.6 Euler Equations

$$
\begin{aligned}
w_0 &= \frac{1}{\rho} \\
\rho_t &= -\rho u_x - \rho_x u \\
u_t &= -p_x w_0 - u_x u \\
p_t &= -u_x p - p_x u \\
w_{0_t} &= w_0 u_x - w_{0_x} u
\end{aligned}
$$

B.7 FitzHugh-Nagamo System

$$
\begin{aligned}
&w_0 = v^2 \\
&v_t = 100\varepsilon^2 v_{xx} + 100r + 100w_0 v + 10v - 110w_0 - 100u \\
&u_t = -\gamma u + hv + r \\
&w_{0_t} = 200\varepsilon^2 \left(-v_x^2 - \frac{w_{0_{xx}}}{2}\right) + 200rv - 220w_0 v - 200vu + 200w_0^2 + 20w_0
\end{aligned}
$$

B.8 Schnakenberg Equations

$$
\begin{aligned}
&w_0 = u^2 \qquad w_1 = vu \\
&u_t = a\gamma + D_u u_{xx} + D_{uv} v_{xx} - \gamma u + \gamma v w_0 \\
&v_t = b\gamma + D_v v_{xx} + D_{vu} u_{xx} - \gamma v w_0 \\
&w_{0_t} = 2a\gamma u + D_u(-2u_x^2 + w_{0_{xx}}) + 2D_{uv} u v_{xx} + 2\gamma w_0 w_1 - 2\gamma w_0 \\
&w_{1_t} = a\gamma v + b\gamma u + D_u(-v_{xx}u - 2u_x v_x + w_{1_{xx}}) + D_{uv} v v_{xx} + D_v u v_{xx} + D_{vu}(-u_x^2 + \frac{w_{0_{xx}}}{2}) \\
&- \gamma uv - \gamma w_0 w_1 + \gamma w_1^2
\end{aligned}
$$

B.9 Brusselator System

$$
\begin{aligned}
&w_0 = u^2 \qquad w_1 = uv \\
&u_t = b\gamma v w_1 + d_1 u_x + \lambda + u(-b\lambda - \lambda) \\
&v_t = a^2\lambda u - a^2\lambda u w_1 + d_2 v_x \\
&w_{0_t} = 2b\lambda w_0 w_1 + 2d_1 u_x u + 2\lambda u + u^2(-2b\lambda - 2\lambda) \\
&w_{1_t} = a^2\lambda u^2 - a^2\lambda w_0 w_1 + b\lambda w_1^2 + d_1 u_x v + d_2 u v_x + \lambda v + w_1(-b\lambda - \lambda)
\end{aligned}
$$

B.10 Allen-Cahn Equation

$$
\begin{aligned}
&w_0 = u^2 \\
&u_t = -uw_0 + u + u_{xx} \\
&w_{0_t} = -2u_x^2 - 2w_0^2 + 2w_0 + w_{0_{xx}}
\end{aligned}
$$

B.11 Heat Equation

The quadratic system below corresponds to the one yielded by setting the equation parameter $p = 6$ and running the Nearest Neighbor search algorithm.

$$
\begin{aligned}
&w_0 = u^5 \qquad w_1 = u_x^2 \qquad w_2 = u^3 \\
&u_t = w_2^2 + u_{xx} \\
&w_{0_t} = 5w_0^2 + w_{0_{xx}} - 20w_1w_2 \\
&w_{1_t} = -2u_{xx}^2 + 12w_0w_1 + w_{1_{xx}} \\
&w_{2_t} = -6w_1u + 3w_0w_2 + w_{2_{xx}}
\end{aligned}
$$

Appendix C

Package Use

The code snippet below shows how to use our software to obtain a quadratization for a given PDE system.

```
# definition of functions and independent variables of the PDE
t, x = symbols('t,x')
u = Function('u')(t,x)

# definition of the PDE
u_t = u**3 * D(u, x, 3)

# function for finding a quadratization
# input: system of equations, number of differentiations for
# the new variables, sorting function
# output: a quadratization set of the system of equations
quadratize(func_eq=[(u, u_t)], n_diff=4, sort_fun=by_fun)
```