DISCOVERY OF EXACT EQUATIONS IN MATHEMATICAL DATABASES

Boštjan Gec

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Dr. Ljupčo Todorovski, Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Co-Supervisor: Prof. Dr. Sašo Džeroski, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Asist. Prof. Dr. Tea Tušar, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Zoran Bosnić, Member, Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia
Assoc. Prof. Dr. Nikola Simidjievski, Member, Institut Polytechnique de Paris France, Paris, France

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL



Boštjan Gec

DISCOVERY OF EXACT EQUATIONS IN MATHEMATICAL DATABASES

Doctoral Dissertation

ODKRIVANJE EKSAKTNIH ENAČB V ZBIRKAH MATEMATIČNIH PODATKOV

Doktorska disertacija

Supervisor: Prof. Dr. Ljupčo Todorovski

Co-Supervisor: Prof. Dr. Sašo Džeroski

Ljubljana, Slovenia, September 2026

Acknowledgments

First, I thank my supervisor Ljupčo Todorovski for guiding me on the path of becoming a researcher and pushing me to reach my limits.

My co-supervisor Sašo Džeroski for giving me the opportunity to work here, providing project ideas, and creating collaborative connections.

The members of the evaluation board: Tea Tušar, Zoran Bosnić, and Nikola Simidjievski for the effort they put into carefully reading this work and providing constructive criticism.

Paul Breiding, Giacomo Maletto, Bernd Sturmfels, Carlos D'Andrea, and Luca Sodomaco for the insightful discussions and valuable feedback during the poster session at MEGA 2024 (Effective Methods in Algebraic Geometry) and in the conversations that followed, which helped shape the subsequent direction of my research.

David E. Narváez for providing rigorous proofs of the two newly discovered equations, which made their discovery one of the most exciting moments of my PhD journey.

My colleagues Jure Brence, Nina Omejc, Sebastian Mežnar, Sintija Stevanoska and Martin Perčinič for our collaboration in the field of equation discovery as well as all the laughter we've enjoyed in the office and all the gossip we've shared at lunch.

The members of the Department of Knowledge Technologies, for helping create an enjoyable and stimulating research environment.

Members of the student council of Jozef Stefan International Postgraduate School for organizing social events.

Most of all, I thank my family and friends for their support.

Finally, I acknowledge the financial support from the Slovenian Research and Innovation Agency (ARIS), primarily through the research project GC-0001 and the core funding P2-0103.

Abstract

Equation discovery, closely related to and frequently discussed under the term symbolic regression, studies methods for identifying symbolic equations that describe relationships in observed data. Most existing approaches to equation discovery are designed for noisy measurements and therefore aim to discover approximate equations. In many mathematical settings, however, approximation is not sufficient: the goal is to recover exact equations from noise-free data. This dissertation introduces exact equation discovery, a form of equation discovery concerned with finding equations that reconstruct the observed data exactly and, where possible, are equivalent to the underlying ground-truth equations.

The dissertation is motivated by the need to discover exact laws in mathematical data, such as the integer sequences from the On-Line Encyclopedia of Integer Sequences, OEIS. Existing approaches to equation discovery typically come either from approximate symbolic regression or from specialized methods in experimental mathematics and computer algebra. This work combines these two types of approaches by developing broadly applicable methods for exact equation discovery grounded in number theory and computational algebraic geometry.

Two algorithms are developed in the dissertation. The first, *Diofantos*, reformulates exact equation discovery over the integers as a problem of solving systems of linear Diophantine equations. By replacing regression-based approximation with exact arithmetic and Diophantine solving, *Diofantos* discovers polynomial equations that match the observed data exactly. The second algorithm, *MoadeeB*, views equation discovery as the computation of the vanishing ideal of the observed data. Using Gröbner bases, *MoadeeB* supports the discovery of exact polynomial, implicit polynomial, and rational equations over integers and rational numbers.

The methods are evaluated on large-scale OEIS benchmarks and synthetic benchmarks generated from polynomial, rational, and implicit polynomial equations. On more than 30,000 OEIS sequences, they reconstruct known relations and discover undocumented ones. They compare competitively with state-of-the-art symbolic regression, program synthesis, and experimental mathematics approaches. Their practical value is further demonstrated by the discovery of one simplified recurrence and two unknown recurrence relations for OEIS sequences, subsequently approved by OEIS curators. The synthetic benchmarks confirm their complementarity: *Diofantos* is best suited to polynomial equations, whereas *MoadeeB* is better suited to rational and implicit equations.

Overall, the dissertation makes three main contributions. First, it defines exact equation discovery as a distinct task at the intersection of equation discovery and experimental mathematics. Second, it introduces two complementary methods for this task. Third, it reformulates the discovery of recurrences from integer sequences as an equation discovery problem, thereby connecting symbolic regression more directly with mathematical databases such as the OEIS. Taken together, the results show that exact equation discovery is feasible and holds promise for reconstructing known mathematical relations, generating new mathematical hypotheses, and enriching mathematical knowledge bases.

Povzetek

Odkrivanje enačb, tesno povezano s simbolno regresijo in pogosto obravnavano pod tem izrazom, preučuje metode za iskanje simbolnih enačb, ki opisujejo zakonitosti v opazovanih podatkih. Večina obstoječih pristopov k odkrivanju enačb je zasnovana za šumne meritve, zato je njihov cilj predvsem odkrivanje približnih enačb. V številnih matematičnih okoljih pa približki niso dovolj; cilj je iz podatkov brez šuma rekonstruirati eksaktne enačbe. V tej disertaciji uvedemo eksaktno odkrivanje enačb, obliko odkrivanja enačb, usmerjeno v iskanje enačb, ki opazovane podatke rekonstruirajo eksaktno (torej brez napak) in so, kadar je to mogoče, ekvivalentne izhodiščnim enačbam.

Izhodišče disertacije je potreba po odkrivanju eksaktnih zakonitosti v matematičnih podatkih, kakršna so celoštevilska zaporedja iz spletne zbirke *On-Line Encyclopedia of Integer Sequences*, OEIS. Obstoječi pristopi k odkrivanju enačb običajno izhajajo iz približne simbolne regresije ali iz specializiranih metod eksperimentalne matematike in računalniške algebre. To delo povezuje obe vrsti pristopov z razvojem širše uporabnih metod za eksaktno odkrivanje enačb, utemeljenih v teoriji števil in računski algebraični geometriji.

V disertaciji razvijemo dva algoritma. Prvi, *Diofantos*, problem eksaktnega odkrivanja enačb nad celimi števili prevede na reševanje sistemov linearnih diofantskih enačb. Pri tem *Diofantos* aproksimativno naravo regresije nadomesti z eksaktno aritmetiko in metodami za reševanje diofantskih enačb ter tako omogoča odkrivanje polinomske enačbe, ki se z opazovanimi podatki ujemajo brez odstopanj. Drugi algoritem, *MoadeeB*, obravnava odkrivanje enačb kot izračun ideala polinomov, ki so ničelni na opazovanih podatkih. Z Gröbnerjevimi bazami *MoadeeB* omogoča odkrivanje eksaktnih polinomskih, implicitnih polinomskih in racionalnih enačb nad celimi in racionalnimi števili.

Metodi ovrednotimo na obsežnih testnih množicah iz OEIS in na sintetičnih testnih množicah, ustvarjenih na podlagi polinomskih, racionalnih in implicitnih polinomskih enačb. Na več kot 30.000 zaporedjih spletne zbirke OEIS rekonstruirata številne znane relacije in odkrijeta tudi nekatere doslej nedokumentirane relacije. Po uspešnosti sta konkurenčni najsodobnejšim pristopom simbolne regresije, sinteze programov in eksperimentalne matematike. Njuno praktično vrednost dodatno pokažemo z odkritjem ene poenostavljene rekurzije in dveh predhodno neznanih rekurzivnih relacij za zaporedja OEIS; uredniki OEIS so te rezultate nato vključili v zbirko. Sintetične preizkusne množice potrdijo, da se metodi dopolnjujeta: *Diofantos* je najprimernejši za polinomske enačbe, *MoadeeB* pa za racionalne in implicitne enačbe.

Disertacija kot celota prinaša tri glavne prispevke. Prvič, eksaktno odkrivanje enačb opredeli kot samostojno nalogo na stičišču odkrivanja enačb in eksperimentalne matematike. Drugič, za to nalogo uvede dve komplementarni metodi. Tretjič, odkrivanje rekurzij iz celoštevilskih zaporedij preoblikuje v problem odkrivanja enačb in s tem simbolno regresijo neposredneje poveže z matematičnimi podatkovnimi zbirkami, kot je OEIS. Rezultati kot celota kažejo, da je eksaktno odkrivanje enačb izvedljiv in obetaven pristop k rekonstrukciji znanih matematičnih relacij, oblikovanju novih matematičnih hipotez ter bogatenju matematičnih baz znanja.

Contents

List of Figures **xv**

List of Tables **xvii**

List of Algorithms **xxi**

Abbreviations **xxiii**

1 Introduction **1**
1.1 Motivation . . . 2
1.1.1 General-purpose equation discovery . . . 2
1.1.2 Limitations of approximate equation discovery . . . 3
1.1.3 Exact equation discovery . . . 3
1.2 Problem Definition . . . 4
1.3 Aim and Objectives of the Dissertation . . . 4
1.4 Hypotheses . . . 5
1.5 Argumentation of the Scientific Relevance . . . 5
1.6 Methodology . . . 6
1.6.1 Design and implementation . . . 6
1.6.2 Empirical evaluation . . . 7
1.7 Organization of the Dissertation . . . 9

2 Related Work **11**
2.1 Automated Discovery in Pure Mathematics . . . 11
2.2 Computational Discovery Approaches Using the OEIS . . . 13
2.3 Existing Work on Equation Discovery . . . 14
2.4 Computing Gröbner Bases from Data . . . 16
2.5 Multivariate Polynomial Interpolation . . . 17
2.6 Algebraic and Computer-Algebra Approaches . . . 17

3 Discovery of Exact Equations via Solving Diophantine Equations **19**
3.1 *Diofantos* . . . 19
3.1.1 Solving diophantine equations . . . 20
3.1.2 Worked example of solving Diophantine equations . . . 21
3.1.3 Construction of higher-degree features . . . 23
3.1.4 Application to integer sequences . . . 24
3.1.5 Computational complexity . . . 25
3.2 Empirical Evaluation . . . 26
3.2.1 Experimental setup . . . 26
3.2.2 Linear recurrence reconstruction on OEIS . . . 29
3.2.3 Exact equation discovery on core OEIS . . . 32
3.2.4 Comparison with state-of-the-art OEIS discovery methods . . . 32

3.2.5 Discussion of results . . . 34
3.3 Discovery of Novel OEIS Recurrences . . . 35

4 Discovery of Exact Equations via Gröbner Basis Computation 37
4.1 *MoadeeB* . . . 37
4.1.1 Ideals, bases, and Gröbner bases . . . 38
4.1.2 The Möller–Buchberger algorithm . . . 40
4.1.3 A worked example of the Möller–Buchberger algorithm . . . 42
4.1.4 Discovery of exact equations via Gröbner bases . . . 43
4.1.5 Time complexity of discovering exact equations via Gröbner bases . . . 44
4.2 Empirical Evaluation . . . 44
4.2.1 Experimental setup . . . 45
4.2.2 Linear recurrence reconstruction on OEIS . . . 46
4.2.3 Exact equation discovery on core OEIS . . . 47
4.2.4 Comparison with state-of-the-art OEIS discovery methods . . . 48
4.2.5 Relation between *MoadeeB* and *Diofantos* . . . 51
4.2.6 Running times . . . 51
4.3 Discovery of Novel OEIS Recurrences . . . 52

5 Discovery From Non-OEIS Data 53
5.1 Utility of Exact Equation Discovery Outside OEIS . . . 53
5.2 Synthetic Non-OEIS Benchmark for Exact Equation Discovery . . . 55
5.2.1 Probabilistic context-free grammars . . . 55
5.2.2 Generating equations and data sets . . . 56
5.2.3 Experimental setup . . . 58
5.2.4 Polynomial equations . . . 59
5.2.5 Rational equations . . . 61
5.2.6 Implicit equations . . . 63
5.2.7 Running times . . . 65
5.2.8 Discussion . . . 66

6 Conclusions 69
6.1 Validity of the Hypotheses Proposed by the Dissertation . . . 69
6.1.1 Hypothesis 1 . . . 70
6.1.2 Hypothesis 2 . . . 70
6.1.3 Hypothesis 3 . . . 70
6.1.4 Hypothesis 4 . . . 71
6.2 Scientific Contributions . . . 72
6.3 Limitations . . . 73
6.4 Further Work . . . 73

Appendix A Pseudocode of *Diofantos* Experimental Setups 77

Appendix B Pseudocode of *MoadeeB* Experimental Setups 79

Appendix C Equivalence of Linear Recurrences Through Generating Functions 81
C.1 Equation Involving the Generating Function . . . 81
C.2 Solving the Equation . . . 83
C.3 Checking the Equivalence . . . 83

References 85

Bibliography **95**

Biography **97**

List of Figures

Figure 4.1: Two approaches to computing a Gröbner basis of an ideal of points. The direct approach uses the Möller–Buchberger algorithm. The alternative, less efficient approach first constructs a basis for the ideal of the union of single points by multiplying basis elements, as described in Section 4.1.1, and then applies the Buchberger algorithm. 38
Figure 4.2: Conceptual view of computing Gröbner bases from data. A finite set of data points can be interpreted as an algebraic variety, whose vanishing ideal is described by a generating set. The Buchberger algorithm then transforms this generating set into a Gröbner basis. 39

Figure 5.1: Number of correctly reconstructed polynomial equations by degree for *MoadeeB*, *Diofantos*, and SINDy. The dashed line shows the size of each degree group. 61
Figure 5.2: Number of correctly reconstructed rational equations by degree for *MoadeeB*, *Diofantos*, and SINDy. The dashed line shows the size of each degree group. 63

List of Tables

Table 1.1: Simple example of discovering an exact integer equation from data. Given observations of the variables $V = \{x_1, x_2, y\}$, the task is to find an equation of the form $y = e(x_1, x_2)$ that determines the target variable y from the remaining variables, $V \setminus \{y\}$. 4
Table 1.2: Overview of the empirical evaluation. The methods and evaluation criteria differ where required by the input representation, hypothesis class, or output type of the compared approaches. 8

Table 3.1: Transformation of the first seven Fibonacci numbers (bottom) into a data table of five observations of four variables for discovering a recursive equation of order two (top). 24
Table 3.2: Performance comparison of *Diofantos* and SINDy on the *linrec* data set. The table reports the number of sequences for which each method discovered an equation identical to the OEIS-reported recurrence, a different valid equation, or no valid equation. 30
Table 3.3: Complexity comparison of equations discovered by *Diofantos* and SINDy on the *linrec* data set. Discovered equations are compared with their corresponding ground-truth equations by recursion order and by the number of nonzero summands on the right-hand side. Each constant summand is counted as one-half. 31
Table 3.4: Complexity distribution of ground-truth equations not reconstructed by *Diofantos* or SINDy. The table separates failed reconstructions according to whether the order of the ground-truth equation is at most 20 or greater than 20. 31
Table 3.5: Performance comparison of *Diofantos* and SINDy on the *core* data set, grouped by OEIS sequence category. 32
Table 3.6: Performance comparison of *Diofantos* and QSynt on the *linrec* and *core* benchmarks. The comparison is based on the number of reconstructed equations. The column labeled Other reports QSynt-discovered programs for sequences not included in the present study. Since 30 core sequences also belong to the *linrec* data set, the QSynt counts do not sum to the expected total. 33
Table 3.7: Performance comparison of *Diofantos* and TM-OEIS on 10,000 OEIS sequences [34]. Performance is evaluated by the predictive accuracy of the discovered equations. The four experimental settings are defined by the number of input sequence elements used for discovery, either 15 or 25, and the number of subsequent elements predicted, either 1 or 10. . . 34
Table 3.8: Running-time comparison of *Diofantos* and SINDy on the *linrec* and *core* data sets. Times are reported in approximate wall-clock units. A denotes the total time required for all experiments, and R denotes the time required for successful reconstructions. 35

Table 4.1: Exact equation discovery data set derived from the first seven elements of the Fibonacci sequence. 39
Table 4.2: Performance comparison of *MoadeeB*, *Diofantos*, and SINDy on the *linrec* data set. For each method, ID denotes sequences for which the discovered equation is identical to the OEIS-reported recurrence, EQUIV denotes equations provably equivalent to the OEIS-reported recurrence, and VALID denotes equations that are correct on all available sequence elements but whose equivalence could not be established. FAIL denotes sequences for which no valid equation was found, while ID+EQUIV reports the number of provably reconstructed OEIS recurrences. 47
Table 4.3: Breakdown of equations classified as VALID in Table 4.2. The table distinguishes between equations that are provably not equivalent to the OEIS-reported linear recurrence and equations whose equivalence could not be verified using the linear-recurrence equivalence test described in Appendix C. 47
Table 4.4: Performance comparison of *MoadeeB*, *Diofantos*, and SINDy on the core OEIS sequences, grouped by sequence category. The table reports the number of successful reconstructions obtained by each method. 48
Table 4.5: Equations reconstructed or discovered by *MoadeeB* for core OEIS sequences on which both *Diofantos* and SINDy failed. The table reports the OEIS identifier, sequence name, and equation discovered by *MoadeeB*. The final two entries correspond to previously undocumented equations that were subsequently proved rigorously and accepted into the OEIS. 49
Table 4.6: Performance comparison of *MoadeeB*, *Diofantos*, Guessing, and QSynt on the *linrec* and *core* data sets. Results are measured by the number of reconstructed equations or valid programs. The entry All† indicates that Guessing is theoretically guaranteed to reconstruct all linear recurrences. The column labeled Other reports QSynt-discovered programs for OEIS sequences not included in either the *linrec* or *core* data set. The totals reported in the QSynt paper do not equal the sum of the *linrec*, *core*, and Other columns because 30 core sequences also appear in the *linrec* data set. Italics indicates the result obtained by the proposed method on the core OEIS sequences. 49
Table 4.7: Performance comparison of *MoadeeB*, *Diofantos*, and TM-OEIS on 10,000 OEIS sequences [34]. Performance is evaluated by the predictive accuracy of the discovered equations. The four experimental settings are defined by the number of input sequence elements used for discovery, either 15 or 25, and the number of subsequent elements predicted, either 1 or 10. Boldface denotes the best-performing method for each experimental scenario. 50
Table 4.8: Running-time comparison of *MoadeeB*, *Diofantos*, and SINDy on the *linrec* and *core* data sets. Times are reported in approximate wall-clock units using days, hours (h), minutes (min), or seconds (s), as appropriate. A denotes the total time required for all experiments, and R denotes the time required for successful reconstructions. 51

Table 5.1: Overview of the non-OEIS mathematical benchmark equations. The fourteen benchmark equations are grouped by data set identifier. For each entry, the table reports the equation name, governing equation, and variables used for equation discovery. 54

Table 5.2: Performance of *MoadeeB*, *Diofantos*, and SINDy on the synthetic polynomial benchmark under different experimental protocols and parameter settings. The table reports the number of reconstructed equations out of 2000, the corresponding success rates, and the approximate wall-clock time required for a sequential run over all data sets. Times are reported in seconds (s), minutes (min), hours (h), or days (d), as appropriate. The best configurations, which are also presented in Table 5.3, are shown in bold. 59
Table 5.3: Summary of the performance of *MoadeeB*, *Diofantos*, and SINDy on the synthetic polynomial benchmark, grouped by the polynomial degree of the ground-truth equation. Success is measured by the number of reconstructed equations equivalent to the ground truth. Only best-performing configurations are used: default settings for *MoadeeB*, a fixed maximal degree of 3 for *Diofantos*, and, for SINDy, a sweep over maximal degrees from 1 to 10. 60
Table 5.4: Selected equations used to generate the synthetic polynomial benchmark data sets. The table reports the equation identifier, the corresponding ground-truth equation, its polynomial degree, and a reconstruction indicator. For the examples shown, all compared methods either succeeded or failed together. 61
Table 5.5: Performance of *MoadeeB*, *Diofantos*, and SINDy on the rational benchmark, grouped by the degree of the corresponding implicit polynomial equation. Success is measured by the number of reconstructed equations equivalent to the ground truth. The final row reports the number of polynomial equations identified from integer-valued data sets. Best-performing configurations identified on the polynomial benchmark are adopted: default settings for *MoadeeB*, a fixed maximal degree of 3 for *Diofantos*, and, for SINDy, a sweep over maximal degrees from 1 to 10. 62
Table 5.6: Selected equations used to generate the rational benchmark data sets. The table reports the equation identifier, the corresponding ground-truth equation, its implicit polynomial degree, and the methods that successfully reconstructed an equivalent equation. 64
Table 5.7: Performance of *MoadeeB* on the implicit benchmark, grouped by polynomial degree. Success is measured by the number of reconstructed equations equivalent to the ground truth. 64
Table 5.8: Implicit equations used to generate the implicit benchmark data sets. The table reports the equation identifier, the corresponding ground-truth equation, its degree, and whether *MoadeeB* successfully reconstructed an equivalent equation. 65

List of Algorithms

Algorithm 3.1: *Diofantos* 23
Algorithm 3.2: Evaluating *Diofantos* on the *linrec* data set 28

Algorithm 4.1: *MoadeeB* — Möller–Buchberger algorithm–based discovery of exact equations 43

Algorithm A.1: Evaluating *Diofantos* on the *core* data set 77
Algorithm A.2: Evaluating *SINDy-tuned* on the *linrec* data set 78
Algorithm A.3: Evaluating *SINDy-tuned* on the *core* data set 78

Algorithm B.1: Evaluating *MoadeeB* on the *linrec* data set 79
Algorithm B.2: Evaluating *MoadeeB* on the *core* data set 80

Abbreviations

OEIS	...	The On-Line Encyclopedia of Integer Sequences
Diofantos	...	Approach based on solvers of Diophantine equations developed in the dissertation
MoadeeB	...	MÖller-Buchberger Algorithm–based Discovery of Exact Equations, approach developed in the dissertation
AI	...	Artificial Intelligence
ED	...	Equation Discovery
SINDy	...	Sparse Identification of Nonlinear Dynamical systems
SSLDE	...	Solver of Systems of Linear Diophantine Equations
ATP	...	Automated Theorem Proving
LLM	...	Large Language Model
TM-OEIS	...	Transformers Model for learning recurrences from integer sequences
QSynt	...	Program synthesis approach of combining tree search with tree neural networks for computing integer sequences
AVI	...	Approximate Vanishing Ideal
MAVI	...	Monomial-Agnostic Vanishing Ideal
Guessing	...	Theory-guided guessing in computer algebra and experimental mathematics
HNF	...	Hermite Normal Form
LLL	...	Lenstra–Lenstra–Lovász lattice basis reduction algorithm
HPC	...	High-Performance Computing

Chapter 1

Introduction

Mathematics is increasingly supported by large, curated repositories that organize mathematical objects together with their properties, invariants, formulas, and known relations. Prominent examples include the On-Line Encyclopedia of Integer Sequences (OEIS) [1], the ATLAS of Finite Groups [2], [3], the House of Graphs [4], and the L-functions and Modular Forms Database [5]. These resources are more than catalogues of individual results. By representing large families of related objects in a structured and often machine-accessible form, they make it possible to compare objects systematically and to search for regularities that may be difficult to recognize from isolated examples.

Recent advances in automated and AI-assisted mathematics demonstrate the potential of this perspective. Learning-based systems have been used to guide formal proof search [6], [7], [8], discover counterexamples to mathematical conjectures [9], evolve new mathematical constructions [10], and use patterns detected in data to guide the formulation of conjectures [11]. In several cases, computationally detected patterns have subsequently led to rigorous mathematical results. Examples include discoveries in knot theory and representation theory [11], as well as the identification and subsequent mathematical analysis of oscillatory patterns in arithmetic data from the L-functions and Modular Forms Database [12], [13], [14]. Such results illustrate how mathematical databases can serve not only as repositories of established knowledge, but also as foundations for new discoveries.

Mathematical knowledge can take the form of definitions, classifications, algorithms, conjectures, proofs, and many other mathematical artefacts. Among these, exact equations constitute a particularly compact, interpretable, and reusable representation of structure. A recurrence relation can characterize an integer sequence, an algebraic identity can connect apparently unrelated quantities, and an equation involving mathematical invariants can reveal a dependency among families of objects. Moreover, the values stored in mathematical databases are frequently exact rather than measured. Relations among them are therefore commonly expected to hold as exact equalities, not merely as approximate numerical trends.

This dissertation investigates the automated discovery of exact equations from mathematical databases and other noise-free mathematical data sets. Its goal is to develop methods that accept exact numerical data, represented as integer or rational sequences and tables, and return symbolic relations that are satisfied exactly by the supplied observations. The intended generality is methodological rather than absolute: the proposed methods are independent of a particular mathematical field or named database, but require the relevant objects to be representable by exact numerical attributes. Within this setting, the dissertation focuses primarily on polynomial relations, including recurrence equations for integer sequences. It thereby places equation discovery at the intersection of automated mathematical discovery, symbolic machine learning, and exact computer algebra.

1.1 Motivation

Many successful computational-discovery systems in pure mathematics are developed in a bottom-up manner. They begin with the objects, representations, and theory of a particular mathematical field and exploit this structure in the design of the corresponding algorithms. For example, the Berlekamp–Massey algorithm [15] and computer-algebra tools such as GFUN [16], *Guess* [17], and `ore_algebra` [18] use the algebraic theory of sequences, recurrences, and generating functions. The approach of Luzhnica and Kohlhase [19] similarly uses closed forms of generating functions to discover relations specifically among OEIS sequences. In graph theory, *AutoGraphiX* exploits graph invariants and extremal graph search to generate and investigate conjectures [20], [21].

Such specialization is a major source of strength. Domain-specific representations can substantially restrict the search space, while the underlying mathematical theory can provide correctness guarantees and efficient algorithms. At the same time, the resulting methods are usually tied to a particular type of object, relation, or database. Applying them to a different mathematical domain commonly requires a new representation, new background theory, or a substantially redesigned discovery procedure. These and other specialized approaches are discussed in more detail in Sections 2.1, 2.2, and 2.6.

This dissertation considers a complementary question: to what extent can exact mathematical relations be discovered by methods that are not constructed around one particular field or repository? Rather than exploiting the full theory of a specific class of mathematical objects, the proposed methods operate on a shared representation—exact numerical tables—and search for relations within a predefined but broadly applicable class of equations.

1.1.1 General-purpose equation discovery

Equation discovery (ED) [22], closely related to and frequently discussed under the term *symbolic regression*, addresses the automated identification of symbolic mathematical models from data. In contrast to conventional regression, where the structure of the model is selected in advance and only its parameters are estimated, equation-discovery methods search for both the symbolic structure of an equation and the values of its parameters. The resulting models may range from algebraic expressions to recurrence relations and systems of differential equations.

General-purpose equation discovery has a long tradition in machine learning and scientific computing. Genetic-programming methods evolve expression trees representing candidate equations [23], [24]; sparse regression methods such as SINDy identify parsimonious equations from libraries of candidate terms [25]; and grammar-guided approaches incorporate syntactic or domain knowledge into the search space [26], [27]. Neural and hybrid approaches learn symbolic representations, impose suitable inductive biases, or combine learned models with search procedures [28], [29], [30], [31], [32], [33]. More recently, transformer models have been learning direct mappings from numerical data to symbolic expressions and recurrence relations [34], [35], [36]. Various approaches to equation discovery and symbolic regression are discussed in more detail in Sections 2.3, 2.4, and 2.5.

The appeal of equation discovery lies in the interpretability of its output. Unlike a black-box predictor, a closed-form equation can be inspected, compared with existing theory, manipulated symbolically, and potentially proved. Its general data-driven formulation therefore makes equation discovery a natural candidate for extracting explicit knowledge from mathematical databases. However, most existing general-purpose methods were developed primarily for empirical scientific data, where observations are real-valued, measurements contain noise, and an approximately accurate model is the intended result.

1.1.2 Limitations of approximate equation discovery

For data obtained through measurement or simulation, approximate equation discovery is both natural and necessary. Such methods typically evaluate candidate equations using a numerical loss function and search for a model that balances a small residual error against low structural complexity. Equality is consequently interpreted relative to a numerical tolerance rather than as an exact algebraic condition.

This formulation creates a mismatch when the input consists of exact mathematical data. A seemingly straightforward adaptation would be to apply an approximate equation-discovery method, round or rationally reconstruct its floating-point coefficients, and then verify the resulting equation using exact arithmetic. Although this procedure can sometimes succeed, it does not turn an approximate search method into an exact one.

First, rounding or rational reconstruction may select an incorrect coefficient, even when the numerical estimate lies close to the desired value. Second, a candidate with a small residual need not correspond to any nearby equation that holds exactly. Third, converting large integers or rational numbers to floating-point values may discard distinctions that are present in the original data. Finally, exact verification can only accept or reject the candidates produced by the numerical search; it cannot recover an exact relation that the approximate objective failed to identify or ranked poorly. Consequently, numerical post-processing provides a useful heuristic, but it offers no general guarantee that exact relations present in the data will be recovered.

1.1.3 Exact equation discovery

Exact equation discovery treats equality as an algebraic condition from the beginning of the discovery process. Computation over the integers or rationals preserves the input values without converting them to floating-point approximations, and the condition that an equation fits the data can be tested without introducing a numerical tolerance. When a candidate equation is constructed and evaluated using exact arithmetic, its satisfaction of every supplied observation can therefore be certified directly.

Exact satisfaction of a finite data set must nevertheless be distinguished from universal mathematical validity. An equation that holds for every available observation may still overfit the data and need not hold for unobserved objects or sequence terms. Exact equation discovery therefore produces relations that are valid on the supplied data and that may serve as mathematical conjectures. Establishing that such a relation holds in general can still require additional validation, theoretical analysis, or proof.

The literature reveals a comparatively sparsely explored intersection of three methodological requirements. The desired method should discover equations that fit exact data with zero residual, should not be tied to one particular mathematical domain or database, and should construct its coefficients using exact computation rather than numerical approximation followed by coefficient correction. Existing exact methods commonly obtain their guarantees through specialization to sequences or another particular mathematical representation, whereas general-purpose symbolic-regression methods are predominantly numerical and approximate.

This dissertation addresses this intersection through two complementary algebraic approaches. The first reduces the discovery of explicit polynomial equations to solving systems of linear Diophantine equations, whereas the second constructs polynomial relations using vanishing ideals and Gröbner bases. Despite their different mathematical foundations, both approaches are designed to recover relations that hold exactly on integer or rational data and can be applied to tabular data as well as to integer sequences represented through consecutive terms. To make this common objective precise, the following

section formalizes exact equation discovery from tabular data and then specializes it to recurrence discovery for integer sequences.

1.2 Problem Definition

Focal to this dissertation is the general task of discovering exact equations from data and its specific instance of discovering recursive equations for integer sequences. The task of discovering exact equations from data can be formalized as follows:

Given observations of the values of the set of rational variables
$V = \{x_1, x_2, \ldots, x_p, y\}$, where y is a designated *target* variable;

Find an equation of the form $y = e(x_1, x_2, \ldots, x_p)$, where e is a mapping $e\colon \mathbb{Q}^p \to \mathbb{Q}$ from the given values of the non-target variables $x_1, x_2, \ldots, x_p$ from $V \setminus \{y\}$ to the value of the target y. Map e should map each vector of observations of the non-target variables to the corresponding observed value of the target y *exactly*, i.e., for each pair of a vector of observations of the non-target variables $\boldsymbol{x}_{\text{obs}} \in \mathbb{Q}^p$ and the corresponding value of the target $y_{obs} \in \mathbb{Q}$, $y_{obs} = e(\boldsymbol{x}_{obs})$.

An example of this task is provided in Table 1.1. The requirement for exact reconstruction of the values of the target variable contrasts with the usual formalization of the equation discovery (and symbolic regression) task. The latter requires that the values of the target y calculated with the equation closely match the observed ones [24], [26]. Just as general-purpose equation-discovery methods usually return a list of candidate equations rather than a single equation, exact equation-discovery methods may also return multiple candidates that match the observed data exactly.

Table 1.1: Simple example of discovering an exact integer equation from data. Given observations of the variables $V = \{x_1, x_2, y\}$, the task is to find an equation of the form $y = e(x_1, x_2)$ that determines the target variable y from the remaining variables, $V \setminus \{y\}$.

Given			**Find**
x_1	x_2	y	
-2	1	0	$y = x_1 + 2x_2$
-1	2	3	
1	-1	-1	
3	0	3	

Discovering exact recursive equations for integer sequences can now be formulated as follows. *Given* the first N elements of an integer sequence $\{a_n\}_{n=0}^{\infty}$, *find* a recursive formula of the form $a_n = f(n, a_{n-1}, a_{n-2}, \ldots, a_{n-o})$ (where $o < N$) that, given the values of the first o elements of the sequence $\{a_n\}_{n=0}^{\infty}$, can be used to recursively calculate the values of the future sequence elements from the values of the previous ones.

1.3 Aim and Objectives of the Dissertation

The aim of this dissertation is to advance equation discovery and symbolic regression by developing methods for discovering exact mathematical equations from data. In contrast to conventional symbolic-regression approaches that typically seek approximate relations,

the proposed methods target exact equations over the integers or rational numbers by exploiting techniques from number theory and computational algebra.

To achieve this aim, the dissertation pursues three closely related objectives: the design, implementation, and empirical evaluation of new methods for exact equation discovery.

The first objective is to develop a formal and computational framework for discovering exact equations. This includes defining the task in a way that distinguishes it from approximate symbolic regression and introducing two complementary approaches. The first reformulates equation discovery as the solution of systems of linear Diophantine equations, enabling the discovery of exact equations with integer coefficients. The second derives polynomial systems from observed data and applies Gröbner-basis computation to discover exact polynomial and rational relations. Particular attention is given to the application of these approaches to integer sequences from the On-Line Encyclopedia of Integer Sequences (OEIS), including the transformation of sequences into suitable tabular representations and the assessment of whether known recurrence relations can be recovered from data.

The second objective is to translate these theoretical frameworks into practical algorithms and reusable software. The resulting methods are implemented in a publicly available, open-source library that provides a common interface for exact equation discovery and serves as the computational basis for the experiments presented in the dissertation. Making the implementations, experimental procedures, and data available also supports reproducibility and facilitates further research in automated mathematical discovery.

The third objective is to evaluate the proposed methods empirically. The evaluation considers both synthetic data and real mathematical data from the OEIS and addresses two complementary questions: whether the methods can reconstruct known equations and whether they can discover exact relations that were not previously documented. The experiments also examine computational efficiency and scalability, as well as the applicability of the methods beyond integer sequences. Together, these evaluations assess the practical usefulness of the proposed approaches as tools for exact equation discovery and automated mathematical exploration.

1.4 Hypotheses

The hypotheses investigated in this dissertation are the following.

- H1 We can design novel frameworks for the discovery of exact equations based on solvers for Diophantine equations and the computation of Gröbner bases.
- H2 We can implement algorithms based on the designed frameworks in a software tool that enables the automated discovery of exact equations from mathematical data.
- H3 The developed algorithms compare favorably with existing approaches in terms of performance and applicability across a substantial subset of exact equation discovery tasks.
- H4 The developed approaches are able to discover new, previously unknown mathematical knowledge in the form of exact equations.

1.5 Argumentation of the Scientific Relevance

This PhD dissertation has advanced the field of equation discovery by developing new methods for identifying exact mathematical equations directly from data. The proposed

approaches integrate established principles from symbolic regression, number theory, and computational algebra.

1. The first major contribution lies in the innovative application of solvers of Diophantine equations to the problem of exact equation discovery. These solvers enable the automated identification of exact equations that inherently hold for data generated within mathematical domains. Because the discovered equations must satisfy the underlying integer constraints by design, they possess intrinsic validity and reliability, thereby enhancing the robustness of the equation discovery process.

2. The second contribution involves employing Gröbner basis computation to enable the discovery of exact polynomial equations. This approach facilitates the derivation of polynomial and rational relationships involving rational coefficients, eliminating the need for manual construction of higher-degree features. Consequently, this method expands the range of problems addressable by equation discovery systems while improving both their computational efficiency and feasibility. By leveraging algebraic methods, the approach ensures mathematically sound results.

3. The final contribution reframes the discovery of recurrence relations in integer sequences as an exact equation discovery task. Within this framework, the algorithms developed in this dissertation were applied to identify new, previously unknown recurrence relations in integer sequences catalogued in the OEIS. Three novel equations were discovered and subsequently proved to hold, demonstrating the capability of the proposed methods to contribute new mathematical knowledge.

1.6 Methodology

The methodology of this dissertation consists of three main stages: design, implementation, and evaluation.

1.6.1 Design and implementation

To address the task defined in Section 1.2, we design a theoretical framework that employs solvers of Diophantine equations and Gröbner bases.

Solvers of systems of linear Diophantine equations (SSLDEs) are employed to discover exact equations in tabular data consisting of integer values. The approach expresses the values in one selected column as linear combinations of the others by solving a system of linear integer equations. In this way, SSLDEs yield integer solutions corresponding to exact linear equations and can be extended to polynomial forms by introducing additional columns representing products of the original variables.

A Gröbner basis can be derived directly from the given data set using established methods [37]. Because it provides polynomial relations that are simple with respect to the chosen monomial ordering, it may contain strong candidate equations.

In the implementation phase, the algorithms developed during the design stage are expressed in detailed pseudocode and accompanied by explanatory descriptions of their key computational steps. An open-source library is then implemented in Python to support experimentation and validation of the proposed methods. The code focuses on clarity and reproducibility rather than performance optimization, serving primarily as a research tool

for testing and demonstrating the feasibility of the SSLDE- and Gröbner-based approaches to exact equation discovery.

1.6.2 Empirical evaluation

The empirical evaluation in this dissertation is designed to assess the proposed methods from several complementary perspectives. Although the individual experiments are presented alongside the corresponding methodological developments in the following chapters, they form a common evaluation framework. This section provides an overview of that framework by summarizing the data sets, compared methods, and criteria used to assess the discovered equations.

The empirical evaluation assesses two complementary capabilities of the proposed equation-discovery methods. First, we evaluate their ability to *reconstruct* known equations from data, providing a quantitative benchmark whenever a suitable ground-truth relation is available. Second, we examine their ability to *discover new* exact equations that are valid on the available data but are not already documented for the corresponding mathematical objects. Reconstruction, therefore, measures recovery of existing mathematical knowledge, whereas the novelty analysis investigates whether the methods can contribute previously unknown relations.

The evaluation uses both integer sequences from the OEIS and general exact tabular data. An integer sequence is converted into a tabular data set by sliding a fixed-size window over its terms, so that each row contains consecutive sequence elements. An equation discovered among the resulting columns consequently corresponds to a recurrence relation for the sequence. Three OEIS data sets are considered. The *linrec* benchmark contains 27,236 sequences with documented linear recurrences with constant coefficients and serves as the principal large-scale reconstruction benchmark. The *core* benchmark contains 164 mathematically significant and more heterogeneous OEIS sequences and is used both for reconstruction and for identifying potentially novel equations. Finally, the comparison on the 10,000-sequence benchmark of d'Ascoli et al. [34] with the TM-OEIS method is performed under its original predictive evaluation protocol.

To assess applicability beyond integer sequences, we additionally consider exact non-OEIS tabular data. These experiments comprise fourteen selected equations across nine data sets motivated by several mathematical domains, as well as synthetic benchmarks for which the generating equations are known. The latter contain 2,000 polynomial equations, 1,000 equations generated from a rational-expression grammar, and 20 genuinely implicit polynomial equations. These data sets enable evaluation of reconstruction in settings that do not rely on the specific representation or algebraic structure of integer sequences.

The choice of compared methods partly determines which data sets and evaluation protocols can be used. The main comparison concerns the two proposed methods, *Diofantos* and *MoadeeB*, together with SINDy [25], a general-purpose sparse regression approach for equation discovery. These methods operate on tabular data and can therefore be compared on both sequence-derived tables and general exact data, subject to their respective hypothesis classes. In particular, the implicit benchmark is evaluated only with *MoadeeB*, which is designed to discover implicit polynomial relations.

For the OEIS experiments, we additionally compare against methods specialized for integer sequences. QSynt [38], TM-OEIS [34], and the computer-algebra Guessing approach implemented in `ore_algebra` [18] provide useful reference points, but are not directly interchangeable with the tabular equation-discovery methods. QSynt returns executable programs rather than equations, TM-OEIS is evaluated through next-term prediction, and Guessing exploits specialized algebraic structure of integer sequences. We therefore retain the evaluation protocols appropriate to these methods, rather than combining their results

with reconstruction rates.

The principal evaluation criterion is the exact reconstruction of a known ground-truth equation. Importantly, reconstruction cannot in general be assessed by comparing the syntactic forms of two equations: mathematically equivalent relations may have substantially different representations. Nor is deciding the equivalence of arbitrary equations computationally feasible in general. The reconstruction criterion must therefore depend on the class of equations under consideration. For explicit synthetic equations, equivalence can be checked symbolically by simplifying the difference between the discovered and ground-truth expressions. For linear recurrence relations, we exploit the fact that equivalence can be tested through their associated rational generating functions. Appendix C describes this procedure and its implementation in detail. This makes the *linrec* benchmark particularly useful, because reconstruction can be evaluated automatically and at scale using a mathematically meaningful equivalence criterion rather than syntactic matching.

Table 1.2: Overview of the empirical evaluation. The methods and evaluation criteria differ where required by the input representation, hypothesis class, or output type of the compared approaches.

Benchmark	**Primary purpose**	**Methods**	**Principal assessment**
linrec(27,236 OEIS sequences)	Recovery of known linear recurrences	*Diofantos*, *MoadeeB*, SINDy; QSynt as an additional reference	Exact validity and reconstruction with respect to documented recurrences
core(164 OEIS sequences)	Equation discovery on heterogeneous, mathematically significant sequences	*Diofantos*, *MoadeeB*, SINDy; QSynt and Guessing as additional references	Exact validity, reconstruction where applicable, and analysis of potentially novel equations
TM-OEIS benchmark (10,000 OEIS sequences)	Comparison with transformer-based sequence discovery	*Diofantos*, *MoadeeB*, TM-OEIS	Next-element and next-ten-elements predictive accuracy under the protocol of [34]
Non-OEIS mathematical examples (14 equations across 9 data sets)	Applicability beyond integer sequences and OEIS	*Diofantos*, *MoadeeB*	Recovery of known exact relations
Synthetic polynomial (2,000 equations)	Large-scale reconstruction from general tabular data	*Diofantos*, *MoadeeB*, SINDy	Equivalence to the generated ground-truth equation
Synthetic rational (1,000 equations)	Reconstruction involving rational expressions and rational-valued data	*MoadeeB*; *Diofantos* and SINDy as additional reference points	Equivalence to the generated ground-truth equation
Synthetic implicit (20 equations)	Discovery of genuinely implicit polynomial relations	*MoadeeB*	Equivalence to the ground-truth relation, supplemented by manual inspection in ambiguous cases

This distinction also separates *reconstruction* from mere validity on the observed data. A discovered equation is considered valid when it holds exactly on the observations used

for validation. In contrast, the reconstruction rate measures the proportion of benchmark instances for which the discovered equation is equivalent to the known ground truth. An equation may therefore fit every available observation without reconstructing the documented relation. Such cases may correspond to alternative consequences of known relations, finite-sample artifacts, or genuinely new equations.

Potential novelty is investigated primarily on the *core* OEIS benchmark. Equations that satisfy the available sequence terms but are not identified as reconstructions of documented relations are first compared with the information already available in the OEIS. Candidate relations that appear to be previously undocumented are then examined manually, and their general validity is established by mathematical proof. Only relations for which such a proof can be obtained are considered for submission to the OEIS curators. Acceptance into the OEIS provides an additional operational indication that the relation represents useful previously undocumented knowledge. The novelty assessment is therefore more qualitative than reconstruction, but it does not rely solely on agreement with a finite-sequence prefix: candidate discoveries undergo independent mathematical verification before they are reported as new relations.

For comparisons that do not admit an equation-equivalence criterion, we use the evaluation protocol native to the corresponding method. QSynt is assessed through the successful generation of sequence terms. At the same time, the comparison with TM-OEIS follows the predictive protocol of [34], in which relations inferred from the first 15 or 25 terms are evaluated on the following one or ten subsequent terms. These measures are reported separately from reconstruction rates because they assess predictive agreement rather than mathematical equivalence to a known relation.

Table 1.2 summarizes the role of the individual benchmarks, the methods considered, and the principal evaluation criteria. Detailed preprocessing procedures, parameter settings, computational limits, and method-specific adaptations are given in the corresponding experimental sections of the thesis chapters. Wherever possible, common settings are retained across experiments. The overview above therefore provides the common conceptual framework for interpreting the empirical results, while the individual experimental sections specify the details necessary to reproduce each comparison.

1.7 Organization of the Dissertation

In this chapter, we introduced the problem of exact equation discovery and briefly outlined existing related work. We identified the challenges in the field, discussed how different approaches address the challenges, and identified the shortcomings of existing work. In light of the identified challenges, we defined the purposes and goals of this work, the hypotheses to be answered and outlined its contributions to science. The remainder of this dissertation is organized as follows.

In Chapter 2, we expand on the overview of related work given in the introduction by examining existing approaches and their suitability for exact equation discovery.

In Chapter 3, we present the algorithm *Diofantos* and show how the problem of exact equation discovery can be reformulated as a problem of solving systems of linear Diophantine equations. We explain why this reformulation is well-suited to the discovery of exact relations over integers and describe the mathematical principles on which the method is based. Next, we evaluate *Diofantos* on a large collection of sequences from the OEIS and compare its performance with state-of-the-art methods.

In Chapter 4, we further develop the framework of exact equation discovery by relating it to the computation of vanishing ideals and Gröbner bases from algebraic geometry. Based on this connection, we introduce the algorithm *MoadeeB*, designed to discover exact

implicit, polynomial, and rational equations over the integers and the rational numbers. We then evaluate *MoadeeB* in a large-scale study on sequences from the OEIS, where we compare it with methods from symbolic regression, program synthesis, and experimental mathematics for reconstructing known symbolic relations and discovering new ones.

In Chapter 5, we broaden the empirical evaluation of the proposed methods beyond the OEIS setting and beyond the discovery of linear recurrences. First, we study a small benchmark of well-known equations from the mathematical literature in order to illustrate the usefulness of the methods on mathematical data outside the OEIS. Second, we introduce a large-scale synthetic benchmark of tabular data generated from polynomial, rational, and implicit equations, which enables a more systematic evaluation of exact recovery of equations equivalent to the ground truth. Using these benchmarks, we compare *Diofantos* and *MoadeeB* on a wider range of equation classes and analyze their strengths and weaknesses in settings that are more general and more demanding than those considered in the previous chapters.

Finally, in Chapter 6, we review and evaluate the performed work in light of the goals and purposes we have set for the dissertation. Next, we assess the dissertation hypotheses and describe the scientific contributions of the dissertation in more detail. We conclude the dissertation by discussing the limitations of our work and possible directions for further work.

In Appendices A and B, we provide full details of the experimental setups used to evaluate *Diofantos*, SINDy, and *MoadeeB* on the *linrec* and *core* data sets, primarily in the form of pseudocode. Appendix C provides background for automatically checking equivalence between linear recurrences.

Chapter 2

Related Work

This chapter surveys related work from six complementary perspectives. The first two sections review research on computational scientific discovery in mathematics, an area that has attracted growing attention in recent years. We then narrow the focus to OEIS-centered discovery approaches that use the OEIS both as a benchmark and as a structured knowledge resource.

The third section turns to traditional equation discovery techniques. Despite their success in other scientific domains, these methods have made limited contributions to mathematics, largely because they emphasize approximate relations rather than the exact identities required for proofs and formal reasoning.

Next, we examine methods for computing Gröbner bases from data—a central component of one of our approaches—distinguishing between exact algebraic techniques and noise-tolerant variants. We then review multivariate polynomial interpolation methods grounded in mathematical theory, which address a related but distinct problem.

Finally, we situate our work within the broader landscape of algebraic and computer algebra approaches to symbolic inference.

2.1 Automated Discovery in Pure Mathematics

Automated computational discovery in pure mathematics is closely related to experimental mathematics, which uses computational power to identify object properties and patterns. The field of automated discoveries in mathematics revolves mainly around automated theorem proving (ATP), as mathematicians' affinity towards theorems, lemmas, corollaries, etc., makes it a natural fit for automation [39]. ATP has seen significant advancements through methods like "hammering", which leverages computational power to supply relevant information to formal provers and complete proofs [6], [8], [40], [41], [42], [43], [44], [45]. Recent work [6], [7], [8] exemplifies this approach by employing large language models (LLMs) similar to ChatGPT to predict and complete proofs effectively. An empirical study by Han et al. [7] demonstrated that one-third of the proofs generated by their LLM-based method were significantly shorter, where some of the proof reductions led to a remarkable order of magnitude decrease in theorem compilation times.

While recent references in ATP demonstrate current progress and innovations, the foundational work began with the Automated Mathematician, introduced by Lenat [46]. This early algorithm starts with a small database of 100 mathematical concepts expressed in first-order logic, which is expanded by generating corresponding examples using heuristic rules. The Automated Mathematician has demonstrated its ability to discover simple set-theoretic notions, such as De Morgan's law and singletons.

Hansen and Caporossi [20], [21] present a system called *AutoGraphiX*, which makes discoveries in the mathematical field of graph theory. Its capabilities include finding extremal graphs for graph invariants, finding graphs that satisfy given constraints, refuting or suggesting conjectures about graph invariants, and suggesting proofs. Examples of the invariants involved include the stability number, diameter, number of edges, number of pending vertices, and radius. It discovered the linear equation $2\alpha - m - n_1 + 2r - D = 0$, where the symbols α, m, n_1, r, and D denote the stability number, number of edges, number of pending vertices, radius, and diameter, respectively. The equation was also later proved. At a higher level, the system uses the so-called Variable Neighborhood Search algorithm, which is a heuristic search method combining local search and random exploration in an increasingly distant neighborhoods. At a lower level, it has three ways of obtaining conjectures. The first is numerical and uses principal component analysis on values of invariants of extremal graphs. The second is geometric, in the sense that it uses convex hulls of invariants to conjecture inequalities. The third is algebraic and consists of building a database of relations for simple classes of graphs.

One recent work applies an equation discovery approach in the mathematical field of graph theory [47]. It is performed on a data set of graphs and their properties. For example, the discovered equations include equations with the isoperimetric number on the left-hand side and combinations of eigenvalues of the adjacency matrix on the right-hand side. Equations involving graph diameter were also discovered. The Ramanujan Machine [48] algorithmically discovered new equations and conjectures related to Polynomial Continued Fractions. These discoveries included continued fraction representations for fundamental constants like π, e, Catalan's constant, and values of the Riemann zeta function. Their method initially generates expressions containing polynomials and other related functions that may include unknown constants. Using the gradient descent algorithm, these expressions are then fitted against a chosen fundamental constant.

Wagner [9] utilizes reinforcement learning to discover counterexamples for conjectures, including open ones, in extremal combinatorics and graph theory. Examples include inequalities involving natural numbers, the eigenvalues of adjacency matrices, conjectures involving natural numbers, and the determinants of distance matrices. Reinforcement learning guides the probability distribution of training examples for a deep neural network, favoring those with higher scores that may indicate counterexamples. Romera-Paredes et al. [10] introduce FunSearch, an LLM-guided evolutionary program search framework that pairs a frozen code LLM with a deterministic evaluator to evolve short programs that solve mathematical or algorithmic problems. Using this setup, they discover new constructions for the cap set problem and constant-weight admissible sets that improve the best-known lower bounds on cap-set capacity, as well as new online bin-packing heuristics that outperform widely used baselines on standard distributions. The work provides a strong demonstration that LLM-guided search combined with rigorous verification is a powerful paradigm for exact symbolic discovery.

Recently, artificial intelligence has begun to play a complementary role in mathematical research by serving not only as a predictive surrogate for expensive computations, but also as a heuristic instrument for hypothesis generation. Early studies primarily used neural networks to predict properties of mathematical objects, thereby enabling the rapid screening of large families of mathematical objects where direct rigorous evaluation is prohibitively costly. Importantly, subsequent work has shown that applying interpretability methods to trained models—for instance, via attribution or other sensitivity analyses—can expose structured dependencies that are difficult to detect by manual inspection alone. In a prominent example, Davies et al. [11] propose a workflow in which supervised learning is used to detect the presence of relationships and attribution methods are then used

to guide the formulation of conjectures. This approach yielded new results in knot theory and a provably correct formula for computing Kazhdan–Lusztig polynomials from the cube decomposition of Bruhat intervals, connected to the combinatorial invariance conjecture for symmetric groups.

Large, structured data sets in algebraic geometry and number theory have become a fertile testbed for such machine-learning-assisted discovery. For instance, a data set of (473,800,776) Calabi–Yau examples with associated Hodge numbers [49] has supported controlled learning experiments. In particular, He and Lee [50] showed that a comparatively simple feed-forward network can distinguish elliptically fibered examples, while He and Lukas [51] demonstrated accurate supervised prediction of Hodge numbers. The empirical success of these studies suggests that additional, previously hidden structure—perhaps even expressible in closed form—may be present. In a related spirit, the success of predicting elliptic curve rank motivated a subsequent principal component analysis, revealing striking oscillatory patterns in averages of Frobenius traces when conditioned on rank and conductor range [12]. The initial observation, enabled by data from the L-functions and Modular Forms Database [5], catalyzed a sequence of subsequent investigations into this oscillatory phenomenon, including the derivation of an explicit analytic expression [13] and the identification of analogous behavior in related arithmetic settings [14].

A complementary perspective is provided by recent work on Fano varieties, where machine learning is used to uncover effective criteria and provable asymptotics from complex invariants. Coates et al. [52] use a feed-forward neural network to predict the dimension of a Fano variety from its quantum period with approximately 98% accuracy. Analysis of this phenomenon informed the formulation and proof of rigorous asymptotic results that recover the dimension from quantum periods. The same authors also train a classifier achieving about 95% accuracy for detecting whether certain toric varieties of Picard rank two are $\mathbb{Q}$-Fano [53]. The ensuing model-guided investigation then leads to a new global combinatorial criterion for terminal singularities together with an explicit algorithm that is empirically fifteen times faster than a naïve alternative. Collectively, these contributions illustrate a coherent pattern across domains: machine learning can function as a scalable exploratory layer that suggests candidate structure, while the final mathematical outcome is secured through rigorous proof and, in some cases, new efficient algorithms.

2.2 Computational Discovery Approaches Using the OEIS

The On-Line Encyclopedia of Integer Sequences (OEIS) has evolved into a large, curated corpus of integer sequences enriched with formulas, programs, and cross-references. Beyond serving as a repository of numerical data, the OEIS has increasingly been used as a structured knowledge resource for automated discovery, enabling conjecture generation, relation mining, and systematic benchmarking of symbolic inference methods.

In addition to addressing OEIS, central to our research work, Colton [54] reviews various experimental mathematics methods and mathematical data resources pertinent to the topic. He presents the PSLQ algorithm, which is reminiscent of solving integer equations and mentions the MBASE project, which aims to create a database of concepts, conjectures, theorems, and proofs. Other databases are also mentioned: the Inverse Symbolic Calculator of 50 million decimal numbers, the GAP library of 60 million groups, the Mizar library of formalized mathematics, MathWorld online encyclopedia, and MathSciNet, which contains reviews of more than 10 thousand mathematical articles and references for over 100 thousand. It mentions the Graffiti program, which conjectured an invalid inequality in graph theory involving variables such as the chromatic number, radius, maximum degree, and frequency of the maximum degree of a graph. The AutoGraphiX program reviewed

above is also mentioned. The HR program is also mentioned, and it forms conjectures based on examples in a given mathematical theory. It discovered some theorems in algebra and group theory regarding anti-associative algebras, quasigroups, and anti-Abelian groups. It was also used for integer sequences. It invented numerous integer sequences interesting enough to be included in the OEIS. It was also used for discovering conjectures about them and for discovering empirical relations between sequences, such as being a subsequence or having no common terms.

Billey and Tenner [55] delve into the subject of mathematical databases and their significance. They extol the OEIS as the superior mathematical database in a specific context. Furthermore, they provide a compelling argument for the motivation and rationale behind enriching this database with discoveries. On the other hand, the approach of Luzhnica and Kohlhase [19] carries out automated discovery on this OEIS database. It discovers relations between integer sequences with the help of closed forms of their generating functions. In more detail, each integer sequence has its generating function that could consist of infinitely many terms. If the closed forms of two sequences can be calculated, then we can compare them, and if they differ by an obvious small change, we have related sequences. They discovered more than 60,000 new relations this way. In the paper, they also explain how they extracted relations and other available information from the database.

More recent work has explored learning recurrence relations and symbolic programs directly from sequence prefixes, bringing machine learning methods into the OEIS discovery setting. D'Ascoli et al. [34] apply transformers to perform the discovery of recursive equations in the OEIS, which is particularly relevant to our work. Throughout this dissertation, we refer to their algorithms collectively as TM-OEIS and describe them in slightly more detail in Section 2.3, together with other equation discovery methods. Gauthier and Urban [38] explored the use of machine learning to generate computer programs that can accurately produce the correct elements of a given integer sequence. These programs were constructed using a simplified domain-specific programming language encompassing basic arithmetic operations (0, +, and *div*) and fundamental programming constructs (variables and loops). They employed tree neural networks to predict the children in the evolution of the programs' abstract tree code, enabling them to refine the programs from simple to more complex forms progressively. Their approach, called QSynt, yielded correct computer programs for 27,987 sequences in the OEIS, a repository containing 351,663 integer sequences at the time of their investigation. These approaches frame OEIS discovery as a sequence-to-structure learning problem and emphasize scalability and generalization across diverse sequence families.

Our approach extends beyond integer sequences to exact equation discovery from noise-free tabular data. Among OEIS-based methods, the work by d'Ascoli et al. [34] is the closest related work. In addition to methodological differences, the approaches differ in declarative bias and evaluation. Our method is restricted to polynomial recurrences, whereas d'Ascoli et al. [34] can discover more general equations from the number-theoretic domain. Their evaluation tests correctness only on the first ten sequence terms and focuses on one-step prediction. In contrast, we validate candidate equations on all available terms and, when possible, check analytical correctness. The approach of d'Ascoli et al. outperforms two built-in functions of the well-known Mathematica software [56].

2.3 Existing Work on Equation Discovery

Equation discovery can be seen as a branch of computational scientific discovery [22], [57]. The pioneering method from the field, BACON [58], employs simple heuristics for building new equation terms based on the correlations among observed variables. These simple

heuristics led to rediscovering versions of the ideal gas law, Kepler's third law of planetary motion, Coulomb's law, Ohm's law, and Galileo's laws for the pendulum and constant acceleration. From then on, a diverse range of approaches has been developed within the field of equation discovery. These approaches can be classified into four categories based on the methodologies and principles used.

The first category, including Lagrange [59] and SINDy [25], employs linear regression over libraries of higher-order terms constructed from the observed variables. In SINDy, these terms are explicit features (e.g. x^2, x/y^3) that define an almost linear equation structure in which the model is assumed to be a linear combination of the constructed features. Parameter estimation is performed via regression [59] and sparse regression [25] methods such as lasso. The latter enforce that only a small subset of coefficients is nonzero, yielding a short and interpretable equation despite the very large underlying feature library. This sparse symbolic regression framework searches a very large space of candidate equations efficiently, but its expressiveness is limited by the assumption of linear combinations of the preconstructed feature library.

The second category of methods employs evolutionary computing [23], and also includes GPDD [60], Eureka [24], CSR [47], and PySR [61]. These methods use evolutionary algorithms to optimize equations by minimizing predefined error metrics. Starting with an initial population of equations, high-scoring equations are retained. New ones are generated through crossover, involving swapping random subtrees of expression trees corresponding to equations. This process effectively explores the search space using genetic programming.

The third category employs background knowledge from the application domain to constrain the search space for candidate equations. Lagramge [26] and ProGED [27] leverage context-free grammars to constrain the search space. ProGED extends this framework by introducing soft constraints through a probabilistic context-free grammar and employing Monte Carlo sampling to efficiently navigate the potentially vast search space. Process-based modeling [62], [63], [64], [65] further advances equation discovery by aligning with the thought process of domain experts. This approach involves defining processes corresponding to equations and potential relationships between them, thereby significantly reducing the exploration space for candidate equations.

Finally, the fourth equation discovery method category employs neural networks and deep learning. EQL [28] uses a shallow neural network, inherently an equation, with activation functions representing operations in equations. Like sparse linear regression, EQL regularizes network weights to produce more parsimonious equations. AI Feynman [30] exemplifies a physics-inspired approach, leveraging symmetries, separability, compositionality, and other simplifying properties inherent in physics. Different approaches, such as DSO [31], combine genetic programming and reinforcement learning to achieve state-of-the-art results. Some methods, such as GVAE [32] and EDHiE [33], use variational autoencoders to learn a latent representation of the candidate-equation search space, enabling equation discovery via optimization in the learned latent space. Furthermore, Refs. [34], [35] train transformer models on data sets consisting of input–output pairs derived from underlying equations, an approach inspired by the work of Valipour et al. [36]. Lastly, also large language models (LLMs) have been applied to the task [10], [66]. LLM-SR [66] is a symbolic regression framework that treats scientific equations as executable programs and uses large language models, combined with evolutionary search, to propose and refine equation skeletons from data while incorporating scientific priors.

Each of these categories offers unique advantages and is suited to different types of problems, highlighting the richness and diversity of the field of equation discovery. Despite progress in equation discovery, its application to pure mathematics still needs to be improved. The approximate nature of the discovered equations often falls short of the rigor

and precision required for mathematical proofs and formalizations. Future research should focus on developing techniques that produce more precise and rigorous equations, enabling them to contribute more meaningfully to automated discovery in pure mathematics.

2.4 Computing Gröbner Bases from Data

A problem closely related to equation discovery is the computation of algebraic varieties and their defining equations from observed data. Breiding et al. [67] studied this problem from a numerical algebraic geometry perspective, focusing on learning algebraic varieties—i.e., sets of polynomial equations—from tabular data and subsequently computing their geometric properties. While their primary emphasis is on numerical techniques, they explicitly recommend the Möller–Buchberger algorithm [37] in settings where exact, noise-free data are available, which directly aligns with the assumptions made in this dissertation.

The Möller–Buchberger algorithm computes Gröbner bases [68], which can be viewed as canonical representations of polynomial ideals associated with finite point sets. Since our approach builds directly on this algorithm, we describe it in detail in Section 4.1.2. Since its introduction, the Möller–Buchberger algorithm has benefited from substantial improvements in efficiency [69], [70] and remains a state-of-the-art method for computing vanishing ideals from exact data. Alternative exact approaches have also been proposed. Farr and Gao [71] introduced a method that is particularly effective when the number of variables is small relative to the number of data points, although their formulation operates over finite fields rather than integers or rational numbers. Just and Stigler [72] proposed an algorithm optimized for settings in which the number of points in the variety is smaller than the ambient dimension, motivated by applications of Gröbner bases to network reconstruction in molecular biology.

Due to their exact algebraic nature, Gröbner-basis-based methods such as the Möller–Buchberger algorithm are not suitable for numerical data affected by noise or finite-precision measurement error. Even small perturbations can lead to overfitting and the appearance of spurious high-degree polynomials. To address this limitation, a parallel line of research has focused on developing approximate variants of vanishing ideal computation that are robust to noise.

A central idea in this approximate setting is to search for polynomials that almost vanish on the data, typically by enforcing constraints of the form $f(x_1, ..., x_n) < \varepsilon$ for a prescribed tolerance. This idea was formalized in the Approximate Vanishing Ideal (AVI) framework introduced by Heldt et al. [73], which has since found widespread use in machine learning. One notable application is Vanishing Component Analysis [74], which constructs new feature representations in a manner analogous to principal component analysis, but grounded in algebraic geometry.

The AVI framework has been further extended and refined, most notably through the Monomial-Agnostic Vanishing Ideal (MAVI) approach [75]. These approximate vanishing ideal methods have been applied to system identification and differential equation discovery [76], [77], where robustness to noise is essential. Although these approaches target equation discovery problems, they are primarily designed for learning systems of differential equations. To the best of our knowledge, AVI-based methods have not been applied to the discovery of algebraic (non-differential) equations.

One notable advantage of MAVI over many symbolic regression techniques is its ability to capture geometric structure and to discover equations in implicit form. Among symbolic regression methods, only a small number of tools—most prominently SINDy-PI [78]—support implicit equation discovery, as the task is typically formulated in explicit form.

In summary, existing work on computing Gröbner bases from data spans both exact

algebraic methods and approximate, noise-tolerant variants. Our work, *MoadeeB*, builds directly on the exact Möller–Buchberger algorithm [70], extending it from a tool in algebraic geometry to a principled algorithm for exact equation discovery. Unlike approximate vanishing ideal approaches such as AVI and MAVI, which trade exactness for robustness, our method preserves exact recovery guarantees by operating entirely in the noise-free setting.

2.5 Multivariate Polynomial Interpolation

Recent work has revisited polynomial interpolation from a learning perspective, particularly in the context of sparse recovery and compressed sensing. In their recent book, Breiding et al. [79, Section *Learning Varieties*] discuss methods for learning algebraic structures from data by exploiting sparsity in underdetermined linear systems. For the identification of integer or rational polynomials, they recommend approaches based on finite-field arithmetic, which provide robustness and computational efficiency.

A central technique in this area is Prony's method [80], which underlies many modern algorithms for sparse multivariate polynomial interpolation [81], [82], [83]. These methods recover polynomials with few nonzero terms from a limited number of evaluations and have been successfully applied to system identification problems [84], a domain closely related to equation discovery. Their efficiency stems from reducing polynomial recovery to solving structured linear systems under sparsity assumptions.

Ben-Or and Tiwari [85] proposed an exact multivariate polynomial interpolation method that operates in the symbolic domain. Such approaches typically construct multivariate Vandermonde matrices from input data points and recover sparse polynomial representations by solving the resulting linear systems exactly. This formulation is reminiscent of feature-based approaches to equation discovery, including SINDy, as well as our earlier work *Diofantos*, which explicitly constructs higher-order monomial features.

Despite conceptual similarities, multivariate polynomial interpolation methods typically rely on restrictive sampling assumptions, such as evaluations over finite fields or prime-power domains, whereas our approach does not. Moreover, these methods largely operate on approximate numerical data rather than the exact data considered here.

2.6 Algebraic and Computer-Algebra Approaches

The automated discovery of symbolic relations from data has a long tradition in experimental mathematics and computer algebra, where the exact, noise-free setting is often referred to as *guessing* recurrences or differential equations from initial terms. In this paradigm, the goal is to recover symbolic descriptions—such as recurrence relations or generating functions—that are consistent with a finite prefix of a sequence, closely matching the setting of integer-sequence inference.

A foundational result in this area is the Berlekamp–Massey algorithm [15], which reconstructs the unique minimal-order linear recurrence satisfied by a sequence, provided sufficiently many terms are available. This algorithm was later generalized by Sakata [86] to multidimensional arrays, yielding minimal sets of linear recurrence relations from finite data and establishing an algebraic viewpoint in which recurrence relations form polynomial ideals and minimal generating sets correspond to Gröbner bases.

Complementary perspectives were developed through Conway's *number wall*, systematically described by Lunnon [87], which provides algebraic criteria for the existence and order of linear recurrences over general integral domains. This framework was later extended to the inference of deterministic context-free Lindenmayer (D0L) systems [88], [89],

which represent a different hypothesis class based on grammar-like substitution rules rather than linear dependencies.

Building on these foundations, subsequent work introduced experimental methods for guessing generating functions from sequence data [90], later systematized in the GFUN package [16]. Advances in algorithmic efficiency were achieved by Beckermann and Labahn [91] through fast methods based on rational interpolation and Hermite–Padé approximation. These ideas were subsequently extended and implemented in several computer algebra systems, including the Axiom system [92], Krattenthaler's RATE package [93], Kauers' *Guess* package [17], and the `ore_algebra` library in SageMath [18]. More recently, Berthomieu and Safey El Din [94] proposed methods for efficiently guessing Gröbner bases of structured ideals of relations. We refer to this family of classical computer-algebra techniques for inferring recurrence relations, generating functions, or algebraic relations from sequence data as Guessing.

While these computer-algebra approaches provide exact symbolic inference with strong mathematical guarantees, they are typically specialized to sequence-based settings and are developed outside the machine learning paradigm. Our work draws on these algebraic foundations—most notably Gröbner bases and polynomial ideals—but integrates them into an algorithmic framework explicitly designed for equation discovery from tabular data, bridging classical computer algebra and symbolic machine learning.

Chapter 3

Discovery of Exact Equations via Solving Diophantine Equations

In this chapter, we present the algorithm *Diofantos* and show how solvers for systems of linear Diophantine equations [95] can be used for exact equation discovery. Because these solvers search for integer solutions, they are well-suited to identifying exact linear relations among variables. Conceptually, solving a linear Diophantine system is analogous to solving a linear system in classical linear algebra, but with the additional requirement that the unknowns remain integral. This requirement makes standard Gaussian elimination unsuitable in general, since its divisive steps typically introduce non-integer values. Diophantine solvers therefore rely on integer-preserving transformations, often driven by greatest common divisor computations and related techniques.

3.1 *Diofantos*

To solve the task of discovering exact equations introduced in Section 1.2, we propose the algorithm *Diofantos*, which leverages solvers for systems of linear Diophantine equations. Recall that

$$V = \{x_1, x_2, \ldots, x_p, y\}$$

denotes the set of variables, where $x_1, \ldots, x_p$ are the non-target variables and y is the designated target variable. The input to *Diofantos* is a matrix of integers

$$M = (m_{ij}) \in \mathbb{Z}^{m \times (p+1)},$$

representing the observed data. Here, m denotes the number of observations and $p+1$ the number of variables in V. Each row of M corresponds to one observation. The entry m_{ij} represents the observed value of x_j for $j \leq p$, while $m_{i,p+1}$ represents the corresponding observed value of the target variable y.

Note that we can rewrite M as a block matrix

$$M = [X|\boldsymbol{y}], \tag{3.1}$$

where X denotes a matrix from $\mathbb{Z}^{m \times p}$, and $\boldsymbol{y}$ denotes a vector from $\mathbb{Z}^m$. The block matrix distinguishes between the vector of observations $\boldsymbol{y}$ of the target variable y and the matrix of observations X of the non-target variables in $V \setminus \{y\}$. Now, our algorithm is aimed at discovering an equation of the form $y = e(x_1, x_2, \ldots, x_p)$, such that

$$e(X) = \boldsymbol{y}, \tag{3.2}$$

where $e(X)$ denotes the application of the mapping e to each row of the matrix X.

In the case of linear equations, the linear mapping $e(X)$ can be rewritten in matrix form as $X \cdot \boldsymbol{c}$, where $\boldsymbol{c}$ is an integer vector in $\mathbb{Z}^p$. To find $\boldsymbol{c}$, if it exists, one should solve the system of linear Diophantine equations

$$X \cdot \boldsymbol{c} = \boldsymbol{y} \tag{3.3}$$

using well-known and already implemented solvers of systems of linear Diophantine equations [95]. If the system has a solution $\boldsymbol{c} = \begin{bmatrix} c_1 & c_2 & \dots & c_p \end{bmatrix}^\top$, we declare the successful discovery of the linear equation

$$y = \sum_{i=1}^{p} c_i x_i. \tag{3.4}$$

Otherwise, we report failure.

3.1.1 Solving diophantine equations

The preceding formulation reduces equation discovery to the solution of an integer linear system. We next briefly describe the main idea behind solvers for such systems. The material in this section is not required for understanding the remainder of the dissertation. Its purpose is to provide a high-level overview of the underlying solution procedure.

Although the system

$$A\boldsymbol{c} = \boldsymbol{b}$$

resembles an ordinary system of linear equations, the restriction $\boldsymbol{c} \in \mathbb{Z}^p$ is essential. A system may have a solution over the rational numbers but no solution over the integers; for example, $2c = 1$ has the rational solution $c = 1/2$, but no integer solution.

A standard exact approach resembles Gaussian elimination, except that division, which may leave the integers, is replaced by integer-preserving operations based on greatest-common-divisor computations [96]. This approach relies on the *Hermite normal form* (HNF), a canonical triangular (or trapezoidal) form for integer matrices. In essence, we transform the original matrix A into HNF by performing operations that preserve the integer span of its columns. These operations consist of swapping two columns, negating a column, and adding an integer multiple of one column to another. Producing zeros above the diagonal in this way can be viewed as a matrix analogue of computing greatest common divisors among the entries of a selected row.

A sequence of these column operations can be represented by a unimodular matrix $U \in \mathbb{Z}^{p \times p}$ such that the resulting Hermite normal form H can be expressed as

$$H = AU.$$

A matrix is unimodular if its determinant is 1 or -1; consequently, both U and U^{-1} have integer entries. After substituting

$$\boldsymbol{c} = U\boldsymbol{z},$$

the original system becomes

$$H\boldsymbol{z} = \boldsymbol{b}.$$

The transformation establishes a one-to-one correspondence between integer vectors $\boldsymbol{c}$ and $\boldsymbol{z}$, and therefore preserves exactly the integer solutions of the original system.

The lower-triangular structure of H makes the transformed system straightforward to solve. In particular, its solution can be determined by consistency and divisibility checks followed by integer forward substitution. If one of the required divisibility conditions

fails, the original system has no integer solution. Otherwise, an integer solution can be constructed, while any free variables describe the remaining solutions.

The implementation used in this work follows the LLL-based HNF approach of Havas et al. [95], which combines HNF computation with ideas from lattice-basis reduction. The LLL algorithm transforms a lattice basis by repeatedly subtracting suitable integer multiples of basis vectors and swapping them, while preserving the lattice they generate. The aim is to obtain shorter and more reduced basis vectors. This property is especially useful for equation discovery, since small coefficients usually yield simpler and more interpretable equations. Instead of applying the LLL algorithm directly to the columns of A, which would generally not produce the triangular structure characteristic of the HNF, the approach of Havas et al. incorporates LLL-style reduction steps into the computation of the HNF. Their algorithm is derived by considering the limiting behaviour of LLL applied to a suitably augmented and weighted matrix as the weighting parameter γ tends to infinity. In this limit, the LLL reduction operations become Euclidean-style operations suitable for HNF computation.

The precise reduction step depends on the current state of the computation. One illustrative case of such a reduction can be described as follows. In the column-oriented notation used here, suppose that columns $\boldsymbol{a}_i$ and $\boldsymbol{a}_k$ have nonzero entries $a_{p,i}$ and $a_{p,k}$ in the current pivot row p. The reduction replaces

$$\boldsymbol{a}_k \leftarrow \boldsymbol{a}_k - q\boldsymbol{a}_i, \qquad q = \left\lfloor \frac{a_{p,k}}{a_{p,i}} \right\rceil,$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. As in LLL reduction, the integer multiplier q is chosen to reduce the magnitude of the corresponding entry. The operation is unimodular and therefore preserves the integer span of the columns. Repeated reductions and column swaps drive the matrix towards its Hermite normal form while controlling the growth of the intermediate coefficients.

Thus, the method can still be viewed as an HNF-based solver. The LLL perspective primarily determines how the unimodular reductions steps used to construct the HNF are selected.

3.1.2 Worked example of solving Diophantine equations

We provide a simple worked example to illustrate the high-level ideas underlying the Havas LLL–HNF algorithm.

Consider the following system of linear Diophantine equations:

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 1 \\ 3 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}. \tag{3.5}$$

We denote this system by $A\boldsymbol{c} = \boldsymbol{b}$, and denote the columns and entries of A by $\boldsymbol{a}_j$ and $a_{i,j}$, respectively. We track the performed column operations using a unimodular matrix U satisfying

$$AU = H,$$

where H denotes the current transformed matrix after each operation. Algorithmically, we can present the steps by:

$$\begin{aligned} U_0 &:= I, H_0 := A, \\ U_i &:= U_{i-1}V_i, \\ H_i &:= H_{i-1}V_i, \end{aligned}$$

where V_i is a column operation such that $AU_i = H_i$ holds for each $i > 0$.

We observe that A contains only two nonzero entries above the diagonal, $a_{1,2}$ and $a_{2,3}$, which prevent it from already being lower triangular. In the column-oriented presentation used here, the algorithm prioritizes the upper rows, and we therefore begin with the first row. To eliminate $a_{1,2}$, the algorithm performs a reduction involving the first two columns, $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$. Following the reduction described in the previous subsection, the first column is subtracted from the second:

$$\boldsymbol{a}_2 \leftarrow \boldsymbol{a}_2 - q\boldsymbol{a}_1, \qquad \text{where } q = \left\lfloor \frac{a_{1,2}}{a_{1,1}} \right\rceil = \left\lfloor \frac{1}{1} \right\rceil = 1.$$

This yields

$$H_1 = \begin{bmatrix} 1 & 0 & 0 \\ 2 & -1 & 1 \\ 3 & -1 & 1 \end{bmatrix}.$$

Applying the same column operation to the identity matrix records this first transformation in

$$U_1 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We can directly verify that $AU_1 = H_1$.

At this point, one could proceed by eliminating the entry $h_{2,3}$ of H_1. However, the LLL component of the method also aims to control the growth of the entries in the intermediate matrices H and U. It is therefore advantageous to reduce the first column by adding two copies of the second column: $\boldsymbol{h}_1 \leftarrow \boldsymbol{h}_1 + 2\boldsymbol{h}_2$. This gives

$$H_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \text{ and } U_2 = \begin{bmatrix} -1 & -1 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The algorithm can now proceed to the second row, since the first row already has the desired form. To eliminate $h_{2,3}$, it considers the second and third columns of H and performs the reduction $\boldsymbol{h}_3 \leftarrow \boldsymbol{h}_3 - q\boldsymbol{h}_2$, where $q = \left\lfloor \frac{1}{-1} \right\rceil = -1$. Equivalently, the second column is added to the third. This gives

$$H_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & -1 & 0 \end{bmatrix} \text{ and } U_3 = \begin{bmatrix} -1 & -1 & -1 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

where H_3 is now in lower-triangular form.

Another characteristic feature of the Havas algorithm is that a zero column of H can still be used to modify the corresponding transformation matrix U. Although such an operation leaves H unchanged, it may reduce the magnitudes of the entries in U. In this example, the entries of U_3 can be reduced by subtracting its last column from its first and second columns. We consequently obtain matrices $H := H_5$ and $U := U_5$ satisfying

$$AU_5 = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 1 \\ 3 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 1 & -1 & 0 \end{bmatrix} = H_5.$$

From this point onward, we solve $H\boldsymbol{z} = \boldsymbol{b}$ for $\boldsymbol{z}$ and recover $\boldsymbol{c}$ through the transformation $\boldsymbol{c} = U\boldsymbol{z}$. Equating $H\boldsymbol{z} = \boldsymbol{b}$ yields $z_1 = 2, -z_2 = 3, z_1 - z_2 = 2 + 3 = 5$, and the

general solution $\boldsymbol{c} = U\boldsymbol{z} = \begin{bmatrix} -z_3 & 2+z_3 & 1+z_3 \end{bmatrix}^\top$. Choosing, for example, $z_3 = -1$ or $z_3 = 0$ gives the two solutions $\boldsymbol{c}_1 = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^\top$ and $\boldsymbol{c}_2 = \begin{bmatrix} 0 & 2 & 1 \end{bmatrix}^\top$. From the perspective of equation discovery, these solutions correspond to the equations $\boldsymbol{b} = \boldsymbol{a_1} + \boldsymbol{a_2}$ and $\boldsymbol{b} = 2\boldsymbol{a}_2 + \boldsymbol{a}_3$.

The system of Diophantine equations was chosen deliberately to illustrate the utility of the method for discovering recurrences in integer sequences. Assigning a_{n-1}, a_{n-2}, and a_{n-3} to the columns of A, and a_n to the vector $\boldsymbol{b}$, the choice $z_3 = -1$ recovers the Fibonacci recurrence $a_n = a_{n-1} + a_{n-2}$. Analogously, choosing $z_3 = 0$ yields $a_n = 2a_{n-2} + a_{n-3}$, which can be readily derived from the Fibonacci recurrence. Further details on the application to integer sequences are presented in Section 3.1.4.

3.1.3 Construction of higher-degree features

To allow for the discovery of non-linear equations with *Diofantos*, we employ the procedure used in the equation discovery algorithm SINDy [25]. To the initial set of non-target variables from $V \setminus \{y\}$, we add non-linear, high-order terms with multiplication up to a user-defined degree $d_{\max}$. Algorithm *Diofantos*, presented in Algorithm 3.1, implements this procedure in lines 4 to 9: the k-multicombination of elements in a set A is a multiset of k not necessarily distinct elements of A. The multiplication of the variables in the multiset leads to a vector that is added to X as a new column (line 7). The newly added vector corresponds to a new non-linear term of degree k. For example, a 3-multicombination $\{x_1, x_2, x_2\}$ corresponds to the third-degree term $x_1 x_2^2$. In the next step, *Diofantos* adds a vector of ones $\mathbf{1}$ as the first, new column of X, to allow discovery of equations with linear intercept (line 10). Finally, *Diofantos* solves the system of linear Diophantine equations $X \cdot \boldsymbol{c} = \boldsymbol{y}$ for the extended matrix X. The algorithm returns the discovered equation, if the solution exists, and reports failure otherwise (lines 11 to 16).

Algorithm 3.1: *Diofantos*

Input: Matrix M of observations of the variables $V = \{x_1, x_2, \ldots, x_p, y\}$; the last column of M corresponds to the target variable y
Input: The maximal degree $d_{\max}$ of the non-linear terms
Output: Equation of the form $y = f(x_1, x_2, \ldots, x_p)$

```
1  M = [X|y]
2  q = 0
3  T = []
4  for k ∈ {1, 2, 3, ..., d_max} do
5  |  for (x_ℓ1, x_ℓ2, ..., x_ℓk) ∈ k-multicombination(V \ {y}) do
6  |  |  q = q + 1
7  |  |  Add a new column t_q to T with element-wise product of the columns
   |  |   corresponding to x_ℓ1, x_ℓ2, ..., x_ℓk
8  |  end
9  end
10 Solve the system of linear Diophantine equations [1|T] · c = y wrt c
11 if solution c = [c_0  c_1  ...  c_q]^⊤ exists then
12 |  return Equation y = c_0 + Σ_{i=1}^{q} c_i t_i
13 end
14 else
15 |  return Failed
16 end
```

To solve the systems of linear Diophantine equations, we employed the Python package *Diophantine* [97]. The package implements an LLL-based algorithm for computing the Hermite normal form [95] and returns a list of solutions to the system. From this list, we select the first solution.

3.1.4 Application to integer sequences

In Section 1.2, we introduced the task of finding a recursive formula that computes the elements of an integer sequence from its first N terms. Here, we reformulate this task as a special case of the general problem of discovering exact equations. In the next two paragraphs, we introduce the transformation and illustrate it by reconstructing a recursive equation for the Fibonacci sequence.

Given the first N elements of the sequence $\{a_n\}$, i.e., $a_0, a_1, a_2, \ldots, a_{N-1}$ and an assumed recursion order o, we introduce $o+2$ variables $V = \{n, a_{n-o}, a_{n-(o-1)}, \ldots, a_{n-1}, a_n\}$, where a_n denotes the target variable. Note that we add the variable n to V to allow for the discovery of zero-order recursive equations (i.e., equations in closed form). In turn, we transform the first N elements of the sequence into $N - o$ observations (rows in the data table). The i-th observation (row) includes the values $i, a_{i-o}, a_{i-(o-1)}, \ldots, a_{i-1}, a_i$, where index i runs from o to $N - 1$.

Table 3.1 illustrates the simple transformation of the sequence of the first seven Fibonacci numbers into a five-row data table for the discovery of recursive equations of order two ($o = 2$). Note that the assumed order of the recursive equation can differ from that of the true recursive equation for the Fibonacci numbers, which is two. As long as the assumed order is higher than the order of the actual equation, the data table includes all the variables necessary for reconstructing it.

Table 3.1: Transformation of the first seven Fibonacci numbers (bottom) into a data table of five observations of four variables for discovering a recursive equation of order two (top).

n	a_{n-2}	a_{n-1}	a_n
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8

The sequence of the first seven Fibonacci numbers, 0, 1, 1, 2, 3, 5, 8.

Let us revisit the Fibonacci sequence example from Table 3.1 and examine how the *Diofantos* algorithm handles it. Let the inputs to the algorithm be: firstly, the matrix M with the set of variables V containing two variables: a_{n-1} and a_{n-2} and secondly, the maximum order $d_{\max} = 2$. During the execution of the *Diofantos* we get to the following system of linear Diophantine equations in its matrix form (note the first matrix column of all ones):

$$\begin{array}{c} 1 \\ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{cccccc} a_{n-2} & a_{n-1} & a_{n-2}^2 & a_{n-2} \cdot a_{n-1} & a_{n-1}^2 \end{array} \\ \begin{bmatrix} 0 & 1 & 0^2 & 0 \cdot 1 & 1^2 \\ 1 & 1 & 1^2 & 1 \cdot 1 & 1^2 \\ 1 & 2 & 1^2 & 1 \cdot 2 & 2^2 \\ 2 & 3 & 2^2 & 2 \cdot 3 & 3^2 \\ 3 & 5 & 3^2 & 3 \cdot 5 & 5^2 \end{bmatrix} \end{array} \cdot \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{array}{c} a_n \\ \begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \\ 8 \end{bmatrix} \end{array}. \tag{3.6}$$

When applying a solver of Diophantine equations to this system, it returns the solution

$$c_0 = 0, c_1 = 1, c_2 = 1, c_3 = 0, c_4 = 0, c_5 = 0, \tag{3.7}$$

which leads to the second-order recursive equation defining the Fibonacci sequence:

$$a_n = 0 \cdot 1 + 1 \cdot a_{n-2} + 1 \cdot a_{n-1} + 0 \cdot a_{n-2}^2 + 0 \cdot a_{n-2}a_{n-1} + 0 \cdot a_{n-1}^2 = a_{n-1} + a_{n-2}. \tag{3.8}$$

We note two important limitations of the *Diofantos* algorithm. First, it has the same limitation as SINDy: it can only discover equations that are linear in constant parameters. The second limitation, related to computational complexity, will be discussed in what follows.

3.1.5 Computational complexity

The procedure for introducing new terms (lines 4–9 in Algorithm 3.1) can lead to a large number of variables in the system of Diophantine equations. The exact number of terms/variables can be calculated as

$$q := 1 + \sum_{k=1}^{d_{\max}} \binom{p+k-1}{k}, \tag{3.9}$$

which belongs to $\mathcal{O}(p^{d_{\max}})$, where p denotes number of variables and $d_{\max}$ a chosen maximal polynomial degree. If we put this in the context of discovering recursive equations of order o and assume that in that case $p = o+1$, the number of variables in the Diophantine equations becomes a member of $\mathcal{O}(o^{d_{\max}})$. In other words, with increasing values of the recursion order o and maximal degree of new terms $d_{\max}$, the number of variables can become prohibitive for efficiently solving the Diophantine equations. In the current implementation of *Diofantos*, we address this problem by gradually increasing the values of o and $d_{\max}$ until we find a solution or exhaust the user-defined limit on the computational resources available.

Furthermore, the critical computational step of *Diofantos* (line 10 in Algorithm 3.1) is solving the system of linear Diophantine equations using the solver from [95]. The latter is based on the LLL algorithm and has a time complexity of $O((q+m)^4 \log(qB))$ arithmetic operations [98], [99], where q denotes the number of variables, m represents the number of equations, and B indicates the maximal absolute value of the coefficients in the system.

Note that in *Diofantos* arithmetic operations with large numbers have a crucial influence on the time complexity. Therefore, it is more appropriate to measure time complexity in terms of the number of bit operations, which is in $O(q^2(q+m)^4 \log^3(qB))$. Moreover, the space complexity is in $O(q^2(q+m)\log(qB))$ bits [95], [98], [100]. Consequently, the bit-wise computational complexities of *Diofantos* are in $O(p^{2d_{\max}}(p^{d_{\max}}+m)^4(d_{\max}\log p + \log B)^3)$ for time and $O(p^{2 \cdot d_{\max}}(p^{d_{\max}}+m)(d_{\max}\log p + \log B))$ for space, respectively.

When discovering recurrence equations of order o, we have $p = o + 1$. Furthermore, we assume that the number of equations satisfies $m = O(N)$, where N is the number

of training sequence elements, and take $B = O(1 + \max_{1 \le i \le N} |a_i|^{d_{\max}})$, where a_i denotes the i-th sequence element. Under these assumptions, the time and space complexities become $O(o^{2d_{\max}}(o^{d_{\max}} + N)^4 d_{\max}^3 \log^3(o(1 + \max_{1 \le i \le N} |a_i|)))$ and $O(o^{2d_{\max}}(o^{d_{\max}} + N) d_{\max} \log(o(1 + \max_{1 \le i \le N} |a_i|)))$, respectively. Thus, the recurrence order o and, in particular, the maximal polynomial degree $d_{\max}$ are the principal factors governing the computational complexity.

The time complexity of *Diofantos* therefore grows rapidly with the number of variables p and, in particular, with the maximal polynomial degree $d_{\max}$. For example, when $p^{d_{\max}}$ dominates the number of equations m, the polynomial part of the worst-case time bound grows as $p^{6d_{\max}}$. In the case of integer sequences, the recurrence order o plays an analogous role, since $p = o + 1$. Consequently, increasing either the recurrence order or the maximal polynomial degree can lead to a substantial increase in the computational cost of *Diofantos*.

3.2 Empirical Evaluation

In this chapter, we evaluate *Diofantos* on the two OEIS benchmarks introduced in Section 1.6.2: *linrec* and *core*, comprising 27,370 distinct sequences in total. These experiments follow the common evaluation framework established there: *linrec* provides a large-scale benchmark for reconstructing known linear recurrences, whereas *core* provides a smaller and more heterogeneous collection on which both reconstruction and the discovery of potentially new equations can be studied. The main comparison is with SINDy [25], while QSynt [38] and TM-OEIS [34] provide additional sequence-specific reference points under their respective evaluation protocols. The following subsection specifies the construction of the two data sets, the compared algorithms and their settings, and the assessment criteria used in these experiments; the subsequent subsections present and discuss the corresponding results.

3.2.1 Experimental setup

Data sets. We evaluate *Diofantos* on two collections of integer sequences from the OEIS [1]. Each sequence is transformed into tabular data by sliding a window over its terms, as described in Section 3.1.4, so that equations discovered among the resulting columns correspond to recurrence relations for the sequence.

The first data set, denoted by *linrec*, contains 27,236 sequences with documented linear recurrences with constant coefficients, obtained from the OEIS index of linear-recursive sequences [101]. For each sequence, this index provides a vector of constant coefficients that determines its recurrence. For example, the vector $(-1, 0, 2) \in \mathbb{Z}^3$ corresponds to $a_n = -a_{n-1} + 2a_{n-3}$. As of 4 March 2023, when we prepared the *linrec* data set, the OEIS index contained 34,371 linear-recursive sequences. We excluded 663 sequences for which coefficient vectors were unavailable, 4 for which they could not be interpreted, 29 whose coefficient representations were impractical to store, and 6,439 for which the supplied vectors were found to be incorrect. The resulting *linrec* data set therefore contains 27,236 sequences. We identified incorrect coefficient vectors for 6,439 sequences through an automated validation procedure using the available sequence elements.

The second data set, denoted by *core*, contains 164 sequences from the OEIS core collection [102], also recreated on 4 March 2023. Core sequences are intended to represent fundamental, widely applicable, and extensively studied integer sequences. They range from trivial sequences, such as the sequence of odd numbers, through simple sequences with well-known formulas, such as the Fibonacci and Catalan numbers, to sequences such as the primes, for which no comparable recurrence of the type targeted here is generally

available.

At the time of data collection, the OEIS contained 178 core sequences. We excluded seven trivial sequences that were not labeled *nice*, including the natural numbers, constant sequences of zeroes and ones, and the negative integers. We further excluded seven sequences carrying the keyword *more*, indicating that additional terms were needed. Thus, the *core* data set corresponded to the OEIS query `keyword:core keyword:nice -keyword:more`. Thirty of the 164 *core* sequences also occur in *linrec*, giving 27,370 distinct sequences across the two benchmarks.

For each sequence, we retained the first 200 terms whenever available. Some core sequences contain fewer than 200 documented terms; in these cases, all available terms were used. The *linrec* data set additionally stores the coefficient vector of the documented recurrence. Sequence terms are used as input to the equation-discovery methods, whereas the reference recurrence is used only for evaluation. Both data sets are publicly available [103]. The implementation of *Diofantos*, the SINDy-based comparison, the experimental scripts, sequence databases, and result files are available in our Zenodo repository [104].

Compared algorithms and their settings. As discussed in Section 3.1.4, *Diofantos* is applied to a sequence by constructing an input matrix whose variables represent selected sequence terms. For a maximal recurrence order o, these variables are drawn from $a_{n-1}, a_{n-2}, \cdots, a_{n-o}$, and, where required, the index n. The search proceeds from simpler to more complex hypotheses by increasing the permitted recurrence order and, for the *core* experiments, the polynomial degree.

For the *linrec* benchmark, the setup is given in Algorithm 3.2. Because the benchmark is restricted to known linear recurrences, the variable n is excluded, and the maximal polynomial degree is fixed at $d_{\max} = 1$. The recurrence order is increased successively from 1 to 20. At each order o, *Diofantos* is applied to the corresponding observation matrix, and only equations satisfying $\text{order}(e) = o$ are accepted.

This restriction is required because an observation matrix constructed for order o begins with the target value a_o. A recurrence of lower order $r < o$ found from this matrix has therefore not been tested on the earlier applicable instances $a_r, \ldots, a_{o-1}$. Without the order constraint, such a relation could fit every row of the constructed matrix while failing on the complete available sequence prefix. If the lower-order recurrence were valid on all available terms, it should already have been discovered in the earlier iteration for order r. For illustration, consider the sequence $1, 15, 30, 45, 60, \ldots$ and suppose that a data matrix for second order recurrence is constructed as $[X|\boldsymbol{y}] = \begin{bmatrix} 1 & 15 & 30 \\ 15 & 30 & 45 \\ 30 & 45 & 60 \end{bmatrix}$, with columns corresponding to a_{n-2}, a_{n-1}, and a_n. The first order recurrence $a_n = a_{n-1} + 15$ fits every row represented in this matrix, but the omitted first applicable instance, $a_1 = a_0 + 15$, would require $15 = 1 + 15$ and is therefore false. Requiring the discovered relation to have order o prevents such spurious acceptance at a higher-order iteration. The search terminates as soon as an equation of the required order is found or after all orders up to 20 have been tested.

Algorithm 3.2: Evaluating *Diofantos* on the *linrec* data set

Input: List A of the first 200 elements of a sequence a_i.

Output: Linear-recursive equation $a_n = c_0 + \sum_{i=1}^{o} c_i a_{n-i}$

```
1 for o ∈ {1, 2, ..., 20} do
2 |   Construct observation matrix [X|y] from the elements of a_i, corresponding to
  |     variables V = {a_{n-1}, a_{n-2}, ..., a_{n-o}, a_n}.
3 |   if Diofantos([X|y], d_max = 1) returns an equation e and order(e) = o then
4 |   |   return e : a_n = c_0 + Σ_{i=1}^{o} c_i a_{n-i}
5 |   end
6 end
7 return Failed
```

The *core* setup, given in Algorithm A.1 in the Appendix A, broadens the hypothesis class in two respects. First, the polynomial degree is increased from 1 to 3, allowing *Diofantos* to discover equations involving quadratic and cubic multivariate terms. Second, the variable n is included, making closed-form equations expressible. Accordingly, the search begins at recurrence order 0, where no lagged sequence terms are present, and only closed-form relations can be found.

For *core*, the maximal recurrence order is reduced from 20 to 10 because the number of candidate polynomial terms grows rapidly with both order and degree. For comparison, degree 1 and order 20, with n included, yield 22 terms: 20 lagged sequence variables, n, and the constant 1. At degree 3 and order 10, the 11 variables consisting of n and 10 lagged terms generate $\binom{11+3-1}{3} + \binom{11+2-1}{2} + 11 + 1 = 364$ monomials of degree at most three. At degree 3 and order 20, this number would increase to 2,024. Restricting the maximum order to 10, therefore, keeps the higher-degree search computationally manageable.

SINDy is evaluated using its Python implementation, `PySINDy` [105]. The corresponding pseudocode is provided in Algorithms A.2 and A.3 in the Appendix A. To make its hypothesis space comparable to that of *Diofantos*, we use a polynomial feature library that introduces nonlinear terms up to the same maximal degree.

Unlike *Diofantos*, SINDy performs floating-point rather than exact arithmetic. We therefore truncate the input sequence whenever the magnitude of the polynomial features would exceed the effective precision available to the computation. Specifically, for a given $d_{\max}$, each sequence is truncated before the first term whose $d_{\max}$-th power exceeds 10^{16} (lines 1 and 2 of Algorithms A.2 and A.3). This is necessary because the polynomial transformation may contain powers up to $d_{\max}$; beyond this range, floating-point arithmetic cannot reliably distinguish the discrepancies relevant to exact equation discovery.

To reduce dependence on a particular SINDy configuration, we vary two hyperparameters. The first is the sparsification threshold, which controls the strength of regularization and hence the number of retained terms. We test 10 equally spaced values in the interval $[0, 0.9]$ (lines 4 and 5). The second is the use of ensembles: we consider both the default non-ensemble setting and ensembles constructed by sampling rows and columns of the observation matrix (lines 5 and 6). This gives $10 \times 2 = 20$ configurations per sequence.

The maximal recurrence order is matched to the corresponding *Diofantos* experiment: 20 for *linrec* and 10 for *core*. We report two SINDy variants. *SINDy-default* uses the default values of the two hyperparameters, whereas *SINDy-tuned* considers all 20 configurations and reports the best resulting outcome for the sequence.

Because SINDy estimates coefficients approximately, every equation returned by SINDy is subsequently re-evaluated using exact integer arithmetic on all applicable sequence terms in the benchmark. Only equations that satisfy this exact validation step, represented by the VALIDATE operation in Algorithms A.2 and A.3, are retained for evaluation.

For both methods, we impose a wall-clock limit of two days for each algorithm–sequence pair on a high-performance computing (HPC) cluster.

Performance assessment. Following the general criteria introduced in Section 1.6.2, we distinguish between *validity on the observed data* and *reconstruction* of a known relation. For each method and benchmark, we report the number or proportion of sequences for which a valid equation is discovered and, where a suitable reference relation is available, the number or proportion for which that relation is reconstructed.

An equation is considered valid on the data if it holds exactly for every applicable instance in the available sequence prefix. Validity alone does not imply reconstruction: a relation may agree with all observed terms without matching the documented reference equation.

For the *linrec* benchmark, reconstruction is assessed by comparing the coefficient vector of the discovered recurrence with the coefficient vector provided by the OEIS index. A recurrence is counted as reconstructed only when these vectors are equal. This criterion is straightforward to apply at scale, but it is stricter in terms of syntax than mathematical equivalence: two recurrences that generate the same sequence may be represented by different coefficient vectors and would therefore not be recognized as equivalent under this measure. As discussed in Section 1.6.2, establishing equivalence between recurrence relations requires a more general procedure; such a procedure, based on rational generating functions for linear recurrences, is introduced and used in the experiments of the following chapter.

For *core*, where the documented relations are heterogeneous and are not represented by a uniform coefficient-vector format, reconstruction is assessed manually by comparing the discovered equations with the formulas and recurrences recorded in the OEIS. Equations that are valid on the available data but do not reconstruct a documented relation are treated as candidates for novelty and analyzed separately, following the procedure outlined in Section 1.6.2. Candidate relations are compared with existing OEIS information, and their general validity is established manually by mathematical proof where possible before previously undocumented relations are considered for submission to the OEIS curators.

3.2.2 Linear recurrence reconstruction on OEIS

Table 3.2 shows the results of the experiments with the *linrec* data set. We can see from the table that our *Diofantos* approach performs the best with the success rate of 90.91% which amounts to 21.00% of a better performance than any of the SINDy experiments are showing. *Diofantos* was also better in discovering equations identical to the ground truth by a considerable margin of 10.14%.

We can see when comparing the results between the two SINDy experiments that SINDy can be noticeably improved if its hyperparameters are fine-tuned by 2.81% for discovering identical equations and by 5.66% for all correct equations. The performance difference between *Diofantos* and SINDy out of the box is therefore even more staggering.

Table 3.3 reports the comparison results between the complexities of the discovered equations and the ground-truth equations. We consider two metrics of the complexity of a recursive equation. The first is the recursion order, i.e., the maximal difference between the index of the sequence element on the left-hand side and the indices of the elements on the right-hand side of the equation. The second metric is the number of nonzero additive terms (summands) on the right-hand side of the recursive equations, with constant terms counted as 0.5 instead of 1.

The first thing we can observe from the table is that more than half of the equations discovered with any of the three algorithms are simpler than the ground-truth ones. Although this might suggest the potential discovery of novel, simpler equations not present

Table 3.2: Performance comparison of *Diofantos* and SINDy on the *linrec* data set. The table reports the number of sequences for which each method discovered an equation identical to the OEIS-reported recurrence, a different valid equation, or no valid equation.

	Success		Fail
Approach	**Identical**	**Different**	
Diofantos	**9206**	**15555**	**2475**
SINDy-default	5678	11822	9736
SINDy-tuned	6445	12595	8196

in the OEIS, a more careful interpretation is required. To gain a deeper understanding of the observed results, we sampled 100 out of the 15,555 equations discovered by *Diofantos*, stratifying the sample by the order of the discovered equation.

We manually inspected the difference between each sampled equation and its corresponding ground truth. Several key conclusions emerged. The observed difference in complexity primarily stems from inherent characteristics of the ground truth and our experimental setup. In particular, the ground-truth equations include only previous sequence elements on the right-hand side, without a constant term. On the other hand, our experimental setup allows for the discovery of equations that include a constant term, which might lead to an exact recursive equation that is simpler than the ground truth. Consider, for example, the recursive equation $a_n = a_{n-1} + 15$, which is of order 1 and involves a constant term. The recursive equation without a constant term, which can be derived by subtracting the equations for a_n and a_{n-1}, is $a_n = 2 \cdot a_{n-1} - a_{n-2}$. Note that this equation has an order of 2. An example of such a sequence would be the multiples of 15, i.e., the sequence starting with $0, 15, 30, 45, 60, \ldots$.

Thus, the discovered equations involving constant terms can be simpler than the recursive ones without constant terms, which we considered as ground truth in our experiments since it was reported in our source [101]. Often, the simpler discovered equation is already documented in other parts of OEIS. Note, however, that in some cases, these simpler equations lead to discoveries of equations not currently documented in OEIS. Indeed, for the sequence A078475, we discovered an equation similar to the ground truth, but with half the order. In particular, the ground-truth equation

$$a_n = 9a_{n-4} - 36a_{n-8} + 84a_{n-12} - 126a_{n-16} + 126a_{n-20} - 84a_{n-24} + 36a_{n-28} - 9a_{n-32} + a_{n-36}$$

has an order of 36, while *Diofantos* discovered the following exact equation of order 18:

$$a_n = -9a_{n-2} - 36a_{n-4} - 84a_{n-6} - 126a_{n-8} - 126a_{n-10} - 84a_{n-12} - 36a_{n-14} - 9a_{n-16} - a_{n-18}.$$

For two sequences in our sample of 100, *Diofantos* discovered equations that overfit the sequence elements used for discovery. In particular, for A011937 and A118576, which have only 36 and 33 elements available for training, respectively, *Diofantos* has discovered highly complex equations with 20 and 17 summands on the right-hand side and constant coefficients with high magnitudes (1000 and 10^{47}, respectively). This contrasts with the ground-truth equations that involve nine and five summands, respectively, with the magnitudes of the constant coefficients being below 10.

Finally, *Diofantos* never discovers an equation of order higher than the ground truth, which is an important and encouraging result. The other two algorithms based on SINDy lead to such discoveries in relatively rare cases: less than 2% for the order metric and less

than 12% for the metric of nonzero summands. These discoveries are a consequence of the fact that SINDy discovers approximate equations.

Table 3.3: Complexity comparison of equations discovered by *Diofantos* and SINDy on the *linrec* data set. Discovered equations are compared with their corresponding ground-truth equations by recursion order and by the number of nonzero summands on the right-hand side. Each constant summand is counted as one-half.

Approach	**Recursion Order**			**Number of Summands**		
	$<$	$=$	$>$	$<$	$=$	$>$
Diofantos	**15555**	**9206**	**0**	**13097**	**9209**	2455
SINDy-default	11688	5690	122	9853	5716	**1931**
SINDy-tuned	12226	6493	321	10353	6579	2108

Table 3.4 reports on the analysis of the cases when the algorithms failed to discover an exact equation (corresponding to the last column of Table 3.2). For each approach, we report the number of ground-truth equations with order up to 20, which, in principle, our algorithms could reconstruct, and the number of ground-truth equations with higher recursion order that are out of reach for our experimental setup. Recall that we limit the order of the discovered equations to 20. We can observe from the table that most of the cases where *Diofantos* failed to reconstruct the equation correspond to ground-truth equations with an order higher than 20. This is another important and promising result for *Diofantos*, where it significantly outperforms approaches based on SINDy.

Table 3.4: Complexity distribution of ground-truth equations not reconstructed by *Diofantos* or SINDy. The table separates failed reconstructions according to whether the order of the ground-truth equation is at most 20 or greater than 20.

Approach	**Recursion Order**	
	$\leq$ **20**	$>$ **20**
Diofantos	**36**	**2439**
SINDy-default	7060	2676
SINDy-tuned	5534	2662

Running both approaches on such a large scale turned out to have side-effect benefits. Even before running the experiments, we found many typos and incorrect entries on the ground-truth source [101]. However, some issues became apparent only after applying both methods, when additional positive results, such as discovering equations simpler than those reported in the source (e.g., for sequence A322829), revealed problems in the source. After some exchange of emails with maintainers, we found out that some of the entries are done in an automated way and, therefore, prone to errors. This sheds an unreliable light on the mentioned OEIS's wiki page, although some mistakes were resolved during the period we performed the experiments. These errors also largely explain the 36 sequences for which *Diofantos* did not reconstruct a linear recurrence of order at most 20. For 33 of these sequences, the reason was an insufficient number of available terms: all have recurrence order 20 but fewer than 18 initial sequence elements. Although we automatically checked all sequences before including them in the *linrec* benchmark, these cases were not detected. For two of the remaining sequences, *Diofantos* did not terminate within the two-day time limit. The final case was the zero sequence, which was discarded by line 3

of our experimental setup in Algorithm 3.2 at every tested order, since the solver always returned the trivial order-0 solution.

3.2.3 Exact equation discovery on core OEIS

Results for *core* data set, as presented in Table 3.5, show only a slight difference in performance between *Diofantos* and SINDy. *Diofantos* discovers one equation more than *SINDy-tuned*, which discovers one more than *SINDy-default*. *Diofantos* discovered an exact equation for the OEIS core sequence A000262, while both approaches based on SINDy failed. This sequence corresponds to the number of "sets of lists", i.e., the number of partitions of $\{1, 2, \ldots, n\}$ into any number of lists, where a list, in this case, means an ordered subset. The recursive equation for this sequence, documented in OEIS and discovered by *Diofantos* is $a_n = (2n-1)a_{n-1} - (n-1)(n-2)a_{n-2}$.

Apart from these successful reconstructions, *Diofantos* outputted seven equations for sequences A000058, A000396, A000612, A000798, A001699, A005588, and A055512 that hold for the selected initial terms but are likely overfitted. All seven sequences have fewer than 20 documented initial terms, which makes overfitting particularly plausible. This is further indicated by the complexity of the produced equations and, in some cases, by extremely large constant coefficients. For four sequences, we verified the predicted next term using the available ground-truth generation code; all four predictions were incorrect. For the remaining three sequences, verification was inconclusive, either because computing the next undocumented terms was prohibitively expensive or because no readily available Python implementation was found online.

The differences between versions *SINDy-default* and *SINDy-tuned* hold only for one sequence, namely A002531, for which *SINDy-default* was unsuccessful.

Table 3.5: Performance comparison of *Diofantos* and SINDy on the *core* data set, grouped by OEIS sequence category.

Approach	**Trivial (4)**	**Simple (66)**	**Other (94)**	**All (164)**
Diofantos	4	**32**	0	**36**
SINDy-default	4	30	0	34
SINDy-tuned	4	31	0	35

Finally, Table 3.5 shows comparative results in three categories of *core* sequences. We can observe that the differences in the approaches hold only for simple equations. Most of the overfitted discoveries by *Diofantos* occurred on core sequences in the last category, which includes many sequences for which exact recursive equations have not yet been found.

3.2.4 Comparison with state-of-the-art OEIS discovery methods

As outlined in Section 1.6.2 and Table 1.2, the evaluation includes additional comparisons with QSynt and TM-OEIS, two methods developed specifically for integer sequences. For these reference methods, we use the results reported in the respective publications. We first compare the performance of *Diofantos* with that of the QSynt algorithm [38] on the *linrec* and *core* benchmarks. QSynt is reviewed in Section 2.2.

Table 3.6 confirms this expectation in only one part. QSynt successfully reconstructed only 52% of the ground-truth equations for the sequences in the *linrec* data set, a success rate notably lower than 91% of *Diofantos* on the same data set. This is because QSynt

constrains its space of candidate programs so that the programs can only include three different constant values of 0, 1, and 2. Even though higher constant values can be obtained by combining these values using mathematical operations such as addition and multiplication, the complexity of the resulting programs makes their reconstruction difficult and, in many cases, impossible. We conjecture that this is the most probable explanation of why *Diofantos* outperformed QSynt on the *linrec* data set.

Table 3.6: Performance comparison of *Diofantos* and QSynt on the *linrec* and *core* benchmarks. The comparison is based on the number of reconstructed equations. The column labeled Other reports QSynt-discovered programs for sequences not included in the present study. Since 30 core sequences also belong to the *linrec* data set, the QSynt counts do not sum to the expected total.

Approach	***linrec***	***core***	**Other**
Diofantos	**24,761**	36	-
QSynt	14,168	**50**	13,798

On the other hand, QSynt achieves a much higher success rate of 30% on the sequences from the *core* data set, where the success rate of *Diofantos* is 22%. In the case of the core OEIS sequences, QSynt's ability to discover computer programs increases its chances of success. Namely, for many integer sequences, one can write a generating computer program, which, due to the high expressive power of the programming languages, can not be written as a recursive equation, which can be discovered using *Diofantos*. Furthermore, QSynt can discover complex expressions, such as $n^{(n-1)}$, which are out of reach for *Diofantos* since its limitation to the space of polynomial expressions. On the other hand, QSynt failed to reconstruct the simple recursive equation of $a_n = 4a_{n-2} - a_{n-4}$ for one of the *core* sequences, which may indicate a challenge for discovering higher-order recursive programs (and therefore, equations). Lastly, it is worth noting that one could question the meaningfulness of some very complex equations reconstructed by QSynt, such as the one for sequence A001764: $(\text{loop}(\lambda(x,y).x*y,(x+x)+x,1)\ \text{div}\ \text{loop}(\lambda(x,y).(x*y)+x, x+x,1))\ \text{div}\ \text{loop}(\lambda(x,y).x*y,x,1)$.

Note that the results of QSynt include almost 14 thousand successful reconstructions for other OEIS sequences [38], which are not included in our study.

Finally, we compare the performance of *Diofantos* with that of TM-OEIS [34], the transformer-based method reviewed in Section 2.2. At the time of the experiments, personal communication with the authors indicated that their model implementation was no longer available. We therefore restrict the comparison to the 10,000-sequence benchmark used in their study and adopt their predictive evaluation protocol, summarized in Section 1.6.2. Since this benchmark is not restricted to sequences with linear recurrences, we use the *Diofantos* experimental configuration for the *core* benchmark (Algorithm A.1).

Table 3.7 reports the results of the comparison. While *Diofantos* outperforms TM-OEIS when predicting the following ten sequence elements (23% vs. 19% when training on 15 elements and 26% vs. 21% when training on 25 elements), it achieves lower accuracy when predicting the next sequence element (29% vs. 33% when training on 15 elements and 29% vs. 35% when training on 25 elements).

This behavior may indicate overfitting of the approximate equations to the training data: the next element can often be predicted correctly even by an overfitted model, whereas such models tend to fail when predicting sequence elements further away from the training data. In contrast, when an exact equation is successfully discovered by *Diofan-*

tos, it is typically robust and remains accurate when predicting an arbitrary number of subsequent sequence elements.

The lower overall accuracy of *Diofantos* can also be attributed to the fact that, when the method fails to discover an exact equation, it produces no predictions for subsequent sequence elements. In our evaluation protocol, such cases are treated as predicting zero correct elements, which prevents them from contributing to the overall accuracy of *Diofantos*. In contrast, approximate models may still provide partially correct predictions and thus contribute positively to the overall accuracy of TM-OEIS.

Table 3.7: Performance comparison of *Diofantos* and TM-OEIS on 10,000 OEIS sequences [34]. Performance is evaluated by the predictive accuracy of the discovered equations. The four experimental settings are defined by the number of input sequence elements used for discovery, either 15 or 25, and the number of subsequent elements predicted, either 1 or 10.

Approach	$n_{input} = 15$		$n_{input} = 25$	
	$n_{pred} = 1$	$n_{pred} = 10$	$n_{pred} = 1$	$n_{pred} = 10$
Diofantos	29.0	**22.6**	28.5	**26.2**
TM-OEIS	**33.4**	19.2	**34.5**	21.3

3.2.5 Discussion of results

As a consequence of the analysis in Subsection 3.1.5, the main limitation of *Diofantos*, besides its polynomial inductive bias, is its computational complexity. Two factors have the most significant impact on the complexity: the degree of the newly introduced terms $d_{\max}$, which we set to 3, and the maximal recursion order $o_{\max}$, which we set to 20 or 10. Some of the failures to reconstruct known OEIS equations are related to these two settings since *Diofantos* was incapable of reconstructing, e.g., fourth-degree polynomials or linear recursive formulas with order higher than 20.

The third important factor is the magnitude of the input data, i.e., the maximal absolute value of the training sequence elements. This factor is encountered only because we aim to discover exact equations, and its impact is two-fold. First, although the potentially large numbers arising during computations based on exact arithmetic constitute a major source of computational cost, the magnitude of the input data has only a limited influence on the complexity of the crucial step of *Diofantos*—solving a system of Diophantine equations—as discussed in Section 3.1.5. Second and more importantly, it also increases the time complexity of the procedure for checking the exact validity of the equations discovered by SINDy or other general algorithms for approximate equation discovery.

We address these issues by carefully selecting the order for increasing the values of the two critical parameters of *Diofantos* the recursion order o and the maximal degree of newly introduced terms $d_{\max}$. See, for example, the differences between Algorithm 3.2, where we only iterate through increasing recursive orders used for invoking *Diofantos*, and Algorithm A.1, where we first iterate through increasing values of the maximal degree, and then, in the inner loop, iterate through increasing recursive orders. The ordering, taken together with the run-time limit for a single experiment, often prevents *Diofantos* and especially SINDy from rediscovering known equations for OEIS sequences.

The empirical running times of the two algorithms on the two benchmarks, presented in Table 3.8, are consistent with the expectations arising from the theoretical complexity analysis. The table reports approximate wall-clock times for experiments executed on HPC clusters. The experiments were run in parallel, with each sequence processed independently

on a separate computational resource. The approximate wall-clock times were estimated by recording the time at which the batch of jobs was submitted to the HPC cluster and comparing it with the creation times of the corresponding output files. It is important to note that the experiments were constrained to a 2-day running time. Therefore, a reported time of 2 days signifies that certain experiments did not terminate within the allotted 2-day limit. Consequently, for *Diofantos*, the reported wall-clock times approximately reflect the maximum runtime required to process an individual sequence, whereas for SINDy they provide a basis for comparing the running times across different experimental configurations. However, wall-clock time is not an appropriate measure of the time complexity of an equation discovery algorithm, as it depends substantially on implementation details, hardware, and the execution environment rather than solely on the underlying algorithm. We therefore report running times only to provide the reader with a general indication of the practical computational cost of our implementations.

Table 3.8: Running-time comparison of *Diofantos* and SINDy on the *linrec* and *core* data sets. Times are reported in approximate wall-clock units. A denotes the total time required for all experiments, and R denotes the time required for successful reconstructions.

Approach	***linrec* [R/A]**	***core* [R/A]**
Diofantos	33 h / 2 days	10 h / 2 days
SINDy-default	**2 h / 2 h**	**8 s** / 2 days
SINDy-tuned	6 h / 6 h	1 h / 2 days

Despite the limitations related to the computational complexity, *Diofantos* shows an ability to reconstruct a significant portion of previously known recursive equations for two important classes of OEIS sequences. Due to its ability to discover exact equations, it outperforms SINDy and other algorithms for discovering approximate equations. It is competitive with the algorithm for discovering computer programs that generate sequence elements. Finally, the discovery of a new, previously undocumented recursive equation for the OEIS sequence A078475 shows the potential of *Diofantos* for new discoveries.

3.3 Discovery of Novel OEIS Recurrences

As discussed in Subsection 3.2.2, we discovered a new recurrence relation (a simplification of a previously known one) for the integer sequence with OEIS identifier A078475 that had not previously been documented in the OEIS. Both *Diofantos* and SINDy independently identified this relation. The newly discovered recurrence

$$a_n = -9a_{n-2} - 36a_{n-4} - 84a_{n-6} - 126a_{n-8} - 126a_{n-10} - 84a_{n-12} - 36a_{n-14} - 9a_{n-16} - a_{n-18},$$

was subsequently submitted to the OEIS and formally accepted into the corresponding entry, A078475 [106].

Similarly, *Diofantos* discovered the relation

$$a_n = 2 \cdot n - a_{n-2} + 2 \cdot a_{n-1} - 1$$

for the OEIS entry A000330 [107]. However, this result is less significant than the previous one, as it can be readily derived from an already known and documented linear recurrence relation.

Chapter 4

Discovery of Exact Equations via Gröbner Basis Computation

We make explicit a close connection between equation discovery (symbolic regression) and algebraic geometry. The link becomes clearer if we phrase equation discovery not only in the usual *explicit* form, but also as the search for *implicit* relations among variables.

Given observations in tabular form $X \in \mathbb{R}^{m\times n}$ for variables $x_1, x_2, \ldots, x_n$, equation discovery is often posed as finding a simple explicit model for a designated target variable, i.e.,

$$x_j = \tilde{E}(x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n). \tag{4.1}$$

In contrast, we may seek the simplest nontrivial relation[1] of the form

$$E(x_1, x_2, \ldots, x_n) = 0, \tag{4.2}$$

which defines an implicit equation.

The explicit formulation (4.1) can be embedded into (4.2) by restricting E to the form $E(x_1, \ldots, x_n) = x_j - \tilde{E}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)$. Although the two viewpoints are not equivalent in general, the implicit formulation is the natural starting point for algebraic-geometric methods.

4.1 *MoadeeB*

In algebraic geometry [108], polynomial equations are studied via the geometry of their common zero sets, called algebraic varieties. These varieties and the ideals (sets of polynomials) associated with them in a ring (an algebraic structure extending the concept of integers) can be infinitely large. However, the elegance of algebraic geometry lies in its ability to simplify even such complex structures. Through the concept of ideal basis—a possibly finite set of polynomials—entire (possibly infinite) ideals can be represented, effectively condensing the infinite into a manageable form. This fundamental process of simplification allows mathematicians to delve into the intricate relationships between ideals and varieties, unlocking the rich landscape of algebraic structures.

In the interest of precision, without going into all the mathematical intricacies [108], we first define the *vanishing ideal*, denoted $I(X)$, corresponding to a subset X of the so-called *affine space* $\mathbb{A}^n = K \times \cdots \times K$ over an algebraically closed field, such as $\mathbb{C}$. The vanishing ideal $I(X)$ is defined as the collection of all multivariate polynomials f on $X_1, ..., X_n$, that

[1] The identically zero expression $E \equiv 0$ is a trivial exact solution with minimal complexity, but it carries no information and is excluded.

is, $f \in K[X_1, ..., X_n]$, which equate to zero (i.e., vanish) at every point $(x_1, ..., x_n)$ within the specified subset $X \subset K^n$. In other words,

$$I(X) := \{f \in K[X_1, ..., X_n] \mid f(x_1, ..., x_n) = 0 \text{ for all } (x_1, ..., x_n) \in X\}.$$

It is worth noting that the set X is called the *affine variety* of the ideal $I(X)$ if it is precisely the set of all common zeros of the elements of $I(X)$. The link between equation discovery and algebraic geometry can be elucidated as follows: According to the definition above, for each element f of the ideal $I(X)$, the equation

$$f(x_1, ..., x_n) = 0$$

is valid for all points $x = (x_1, ..., x_n)$ in X. Consequently, if the set X is viewed as the input data set and only polynomial expressions are taken into account, the equation discovery problem (eq. 4.2) can be reinterpreted as the task of identifying the simplest element of the associated vanishing ideal. While the scope of equation discovery is not confined solely to polynomials, the utilization of algebraic geometry tools is still a viable approach, particularly in the absence of any background knowledge or constraints on the data.

The simplification of the vanishing ideal, as previously discussed, involves identifying a finite set of polynomials that not only form a basis for the ideal but also satisfy specific conditions. A prime example of such a basis is the *Gröbner basis* [68]. This basis is characterized by additional conditions, defined by a so-called monomial order, which essentially necessitates the lowest degrees of the polynomials in a basis. Calculating Gröbner bases is a well-established problem in commutative algebra, typically resolved using *Buchberger's algorithm* [68]. Given the uniqueness of the Gröbner basis (subject to the choice of a monomial order), it emerges as a natural solution to the equation discovery problem.

Note finally that the Buchberger algorithm calculates a minimal Gröbner basis from a given set of polynomials. To calculate it for a set of points lying in the corresponding variety, one could simply apply the Möller–Buchberger algorithm (Figure 4.1). However, to avoid going too deep into the technical details behind the related algorithms, we will illustrate the basic idea behind them in the next two sections. Rigorous proofs and arguments are left to the reader, while we recommend the book [109] for an introduction to the relevant basic notions in commutative algebra. Understanding these mathematical and technical details is not essential to comprehend the rest of the dissertation.

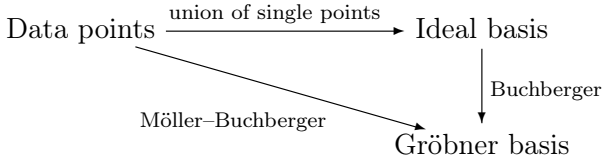


Figure 4.1: Two approaches to computing a Gröbner basis of an ideal of points. The direct approach uses the Möller–Buchberger algorithm. The alternative, less efficient approach first constructs a basis for the ideal of the union of single points by multiplying basis elements, as described in Section 4.1.1, and then applies the Buchberger algorithm.

4.1.1 Ideals, bases, and Gröbner bases

Let us illustrate the use of Gröbner bases for the discovery of the well-known Fibonacci formula $a_n = a_{n-1} + a_{n-2}$ from the first seven elements of the Fibonacci sequence, i.e.,

0, 1, 1, 2, 3, 5, 8. We first transform the sequence into an equation discovery data set corresponding to the variables a_n, a_{n-1}, and a_{n-2}, following the transformation procedure described in Section 3.1.4, to obtain Table 4.1.

Table 4.1: Exact equation discovery data set derived from the first seven elements of the Fibonacci sequence.

a_n	a_{n-1}	a_{n-2}
1	1	0
2	1	1
3	2	1
5	3	2
8	5	3

Our illustration follows the indirect path with two arrows shown in Figure 4.1. The first arrow, which is horizontal, goes from data points to an ideal basis, and the second arrow, which is vertical, goes from an ideal basis to a Gröbner basis. The big picture of it is shown in Figure 4.2. In the context of our example, varieties correspond to sets of data points (rows in the data table), while ideals lie in the ring of polynomials over the variables corresponding to columns of the data table, i.e. a_n, a_{n-1}, and a_{n-2}. Despite the fact that $\mathbb{Q}$ is not an algebraically closed field, the theory still works, so we can consider polynomials over rational numbers, i.e., polynomials from $\mathbb{Q}[a_n, a_{n-1}, a_{n-2}]$. As was mentioned before, an ideal basis represents the whole ideal. To be more specific, each element a from an ideal I can be expressed as a finite linear combination $a = \sum_{i=1}^{m} p_i \cdot b_i$ of basis elements $b_i \in B$, where coefficients $p_i \in \mathbb{Q}[a_n, a_{n-1}, a_{n-2}]$ come from the underlying polynomial ring. The standard notation for an ideal with a finite basis $\{b_1, \cdots, b_m\}$ is $I = \langle b_1, \cdots, b_m \rangle$.

$$\text{Data points} \xrightarrow[\text{singletons}]{\text{union of}} \text{Variety} \xrightarrow[\text{correspond.}]{\text{ideal–variety}} \text{Ideal} \xrightarrow[\text{multiply}]{\text{pairwise}} \text{Basis} \xrightarrow{\text{simplify}} \text{Gröbner basis}$$

Figure 4.2: Conceptual view of computing Gröbner bases from data. A finite set of data points can be interpreted as an algebraic variety, whose vanishing ideal is described by a generating set. The Buchberger algorithm then transforms this generating set into a Gröbner basis.

We can proceed with calculating a Gröbner basis from data points by first considering each point separately. This is due to the fact that a basis of the ideal of the union of two affine varieties can be obtained by pairwise multiplications of the polynomials in the bases of the ideals of the individual varieties [108, p. 24, Section 2.1, Chapter 1]. Note also that a single data point is itself a variety [110, Example 1.5a, p. 7, Chapter 1]. Hence, one could inductively build the entire ideal of a finite number of points by taking unions of varieties. Finally, we can apply Buchberger's algorithm to the resulting ideal to compute a Gröbner basis of the variety corresponding to a finite set of points.

Considering the first row as the variety $\{(1, 1, 0)\}$, we trivially derive that the corresponding ideal I_1 has polynomials $a_n - 1$, $a_{n-1} - 1$, and $a_{n-2} - 0$ for its basis: $I_1 = \langle a_n - 1, a_{n-1} - 1, a_{n-2} \rangle$.

Let us now consider $I_2 = \langle a_n - 2, a_{n-1} - 1, a_{n-2} - 1 \rangle$, a basis of the vanishing ideal corresponding to the second data point $(2, 1, 1)$. For the ideal corresponding to the union

of those two affine varieties of single points, we need to pairwise multiply the polynomials in the corresponding ideal bases, obtaining the following nine polynomials [108]:

$$
\begin{aligned}
I_{12} = \langle & \\
& (a_n - 1)(a_n - 2), && (a_n - 1)(a_{n-1} - 1), && (a_n - 1)(a_{n-2} - 1), \\
& (a_{n-1} - 1)(a_n - 2), && (a_{n-1} - 1)(a_{n-1} - 1), && (a_{n-1} - 1)(a_{n-2} - 1), \\
& a_{n-2}(a_n - 2), && a_{n-2}(a_{n-1} - 1), && a_{n-2}(a_{n-2} - 1) \\
\rangle = \langle & \\
& a_n^2 - 3a_n + 2, && a_n a_{n-1} - a_n - a_{n-1} + 1, && a_n a_{n-2} - a_n - a_{n-2} + 1, \\
& a_n a_{n-1} - a_n - 2a_{n-1} + 2, && a_{n-1}^2 - 2a_{n-1} + 1, && a_{n-1} a_{n-2} - a_{n-1} - a_{n-2} + 1, \\
& a_n a_{n-2} - 2a_{n-2}, && a_{n-1} a_{n-2} - a_{n-2}, && a_{n-2}^2 - a_{n-2} \\
\rangle. &
\end{aligned}
$$

To obtain a Gröbner basis of this ideal, we apply simple transformations of its elements that follow from the axioms for an ideal [109]. For example, by subtracting the sixth polynomial from the eighth one, we obtain $q_1 = a_{n-1} - 1 \in I_{12}$. Furthermore, let $q_2 = a_n \cdot q_1 = a_n a_{n-1} - a_n \in I_{12}$ and $q_3 = a_{n-2} q_1 = a_{n-2} a_{n-1} - a_{n-2} \in I_{12}$. Note that due to the fact that q_3 equals the eighth polynomial above, we can replace q_3 with q_1 in the basis. Similarly, we can further update the basis by removing the second, fourth, fifth, and sixth polynomials, which are equal to $q_2 - q_1$, $q_2 - 2q_1$, q_1^2, and $q_3 - q_1$, respectively, which leads to $I_{12} = \langle a_n^2 - 3a_n + 2, a_n a_{n-2} - a_n - a_{n-2} + 1, a_n a_{n-2} - 2a_{n-2}, a_{n-1} - 1, a_{n-2}^2 - a_{n-2} \rangle$. With a few more similar simplification steps, we can arrive at a Gröbner basis computed by the Buchberger algorithm from the original basis with the nine polynomials:

$$I_{12} = \langle a_n^2 - 3a_n + 2, a_{n-1} - 1, -a_n + a_{n-2} + 1 \rangle.$$

By incorporating the third data point into the variety, we again get nine polynomials by pairwise multiplications of the three polynomials from I_{12} with the three polynomials of the ideal basis corresponding to the third data point $(3, 2, 1)$. This initial basis of nine polynomials simplifies to the Gröbner basis:

$$I_{123} = \langle a_n^2 - 3a_n - 2a_{n-1} + 4, a_n a_{n-1} - a_n - 3a_{n-1} + 3, a_{n-1}^2 - 3a_{n-1} + 2, a_n - a_{n-1} - a_{n-2} \rangle$$

consisting of four polynomials. The fourth one, $a_n - a_{n-1} - a_{n-2}$, corresponds to the Fibonacci formula that generates the considered data points.

In sum, we were able to reconstruct the Fibonacci formula by utilizing only the first three rows of our data set. However, this is a single polynomial among many that could have appeared in a Gröbner basis. If we have calculated a Gröbner basis from more than three points, we might end up with more than four highly complex polynomials. Following Occam's razor, we chose the simplest among the polynomials from a Gröbner basis.

Note, finally, that the described procedure primarily serves illustrative purposes; we use it here to establish the relation between the computation of Gröbner bases and the task of exact equation discovery. In practice, the presented procedure is inefficient, and more effective solutions, such as the Möller–Buchberger algorithm, are used to compute Gröbner bases as depicted in Figure 4.1.

4.1.2 The Möller–Buchberger algorithm

In the previous section, we presented an intuitive explanation of our approach to exact equation discovery. For completeness, we now briefly outline the underlying Möller–Buchberger algorithm, which directly computes a Gröbner basis of the vanishing ideal of a finite set of points. Our aim is neither to provide a formal proof of correctness nor to

present the algorithm in full detail. Instead, we focus on the main ideas underlying its construction of polynomials that vanish on the given data, thereby motivating its use for equation discovery.

The high-level idea underlying the classical Möller–Buchberger algorithm is to interpret monomials as evaluation vectors over the input points and to use linear algebra to detect algebraic dependencies among them. Each detected linear dependence gives rise to a polynomial that vanishes on all input points. The systematic organization of these dependencies yields a Gröbner basis of the vanishing ideal.

The algebraic setting is as follows. Let

$$X = \{P_1, \dots, P_l\} \subset \mathbb{Q}^n$$

be a finite set of distinct points, and consider the polynomial ring

$$R = \mathbb{Q}[x_1, \dots, x_n]$$

equipped with a fixed monomial ordering (for example, the graded lexicographic ordering

$$\{1, x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, x_3^2, x_1^3, \dots\}$$

in the case $n = 3$). For a monomial $m \in R$, we define its evaluation vector on X by

$$m(X) := \begin{bmatrix} m(P_1) & \dots & m(P_l) \end{bmatrix}^\top \in \mathbb{Q}^l.$$

Thus, each monomial is identified with a vector in $\mathbb{Q}^l$.

The algorithm first constructs the staircase (the set of standard monomials) B by processing monomials in increasing order and including a monomial whenever its evaluation vector is linearly independent of those already collected.

We maintain two sets:

- B, a set of monomials,
- $V_B := \{b(X) \mid b \in B\}$, the corresponding set of evaluation vectors,
- G, a set of candidate Gröbner-basis polynomials.

The construction proceeds as follows:

1. Initialize $B := \{1\}$. Since $1(X) = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^\top \neq 0$, this vector is linearly independent.
2. For each subsequent monomial m, compute its evaluation vector $m(X)$.
3. If $m(X)$ is linearly independent of V_B, add it to it and m to B.
4. Otherwise, there exist coefficients $c_b \in \mathbb{Q}$ such that

$$m(X) = \sum_{b \in B} c_b \, b(X).$$

In this case, define

$$g := m - \sum_{b \in B} c_b \, b$$

and add g to the set G.

The vanishing property follows immediately from the linear dependence relation. If

$$m(X) = \sum_{b \in B} c_b \, b(X),$$

then for every $P_i \in X$ we have

$$m(P_i) = \sum_{b \in B} c_b \, b(P_i).$$

Consequently,

$$g(P_i) = 0 \qquad \text{for all } i = 1, \ldots, l.$$

Thus, each detected linear dependence produces a polynomial g that belongs to the vanishing ideal

$$I(X) := \{f \in R \mid f(P_i) = 0 \text{ for all } i\}.$$

This explains the connection between the algorithm and exact equation discovery: linear algebraic dependencies among evaluation vectors translate directly into polynomial equations satisfied by the data.

Since all evaluation vectors lie in $\mathbb{Q}^l$, at most l of them can be linearly independent. Consequently, the construction of B terminates[2] once $|B| = l$, at which point the vectors $\{b(X) \mid b \in B\}$ form a basis of $\mathbb{Q}^l$. From this point onward, every further monomial evaluation must be linearly dependent on V_B and therefore produces a new polynomial in $I(X)$.

The resulting set B is called the *staircase*, *standard monomials*, or *monomial basis* associated with the ideal.

After B has been constructed, the algorithm derives additional relations from the border of the staircase,

$$\partial B := \{x_i b \mid b \in B,\ i = 1, \ldots, n\} \setminus B.$$

Each border monomial $m \in \partial B$ has an evaluation vector $m(X)$ that is necessarily linearly dependent on V_B. Therefore, for each such monomial, we obtain a polynomial relation of the form

$$m - \sum_{b \in B} c_b \, b,$$

which again vanishes on all points of X. Collecting these relations while discarding those that are redundant with respect to the monomials already present in G yields a finite set G.

The resulting set G forms a Gröbner basis of the vanishing ideal. A proof of this fact can be found in the original work of Möller and Buchberger [37] and subsequent refinements [69], [70], [72]. Here, we emphasize its interpretation within exact equation discovery: the algorithm systematically converts linear dependencies among monomial evaluations into algebraic relations satisfied by the data.

4.1.3 A worked example of the Möller–Buchberger algorithm

We provide a simple worked example to make the high-level ideas underlying the Möller–Buchberger algorithm more tangible.

Assume that the first three rows of Table 4.1 are given as input. Equivalently, X consists of the points $(1, 1, 0)$, $(2, 1, 1)$, and $(3, 2, 1)$ in $\mathbb{Q}^3$. Following the outline above, we first

[2] For l distinct points, the evaluation map from polynomials to $\mathbb{Q}^l$ is surjective, and hence the evaluation vectors of monomials span $\mathbb{Q}^l$. Formal proofs of this result and of the termination of the Möller–Buchberger algorithm can be found in the literature cited at the end of this subsection.

work in $R = \mathbb{Q}[a_n, a_{n-1}, a_{n-2}]$ with the graded lexicographic ordering $\{1, a_n, a_{n-1}, a_{n-2}, a_n^2, a_n a_{n-1}, \dots\}$.

Next, we initialize $B := \{1\}$ and $V_B := \{\begin{bmatrix}1 & 1 & 1\end{bmatrix}^\top\}$. We then consider the monomial $m = a_n$ and its evaluation vector $m(X) = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top$, which is linearly independent of $1(X)$. Hence, we obtain $B = \{1, a_n\}$ and $V_B = \{\begin{bmatrix}1 & 1 & 1\end{bmatrix}^\top, \begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top\}$. The next monomial is $m = a_{n-1}$, with $a_{n-1}(X) = \begin{bmatrix}1 & 1 & 2\end{bmatrix}^\top$. This vector is again independent of $1(X)$ and $a_n(X)$, so we get $B = \{1, a_n, a_{n-1}\}$ and $V_B = \{\begin{bmatrix}1 & 1 & 1\end{bmatrix}^\top, \begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top, \begin{bmatrix}1 & 1 & 2\end{bmatrix}^\top\}$. At this point, the stopping criterion $|B| = 3$ is satisfied, since the number of elements of the staircase equals the number of variables. Therefore, V_B forms a basis of the corresponding vector space.

To obtain a Gröbner basis, we proceed by constructing the border of the staircase, $\partial B = \{a_{n-i} b \mid b \in B, i = 0, 1, 2\} \setminus B = \{a_n^2, a_n a_{n-1}, a_{n-1}^2, a_{n-2}, a_n a_{n-2}, a_{n-1} a_{n-2}\}$. The last two monomials in this set are redundant, since they are multiples of a_{n-2}, whose corresponding relation is added to G when a_{n-2} is processed earlier in the border of the staircase. It remains to compute the relations corresponding to these relevant monomials. We can express $a_n^2(X) = \begin{bmatrix}1 & 4 & 9\end{bmatrix}^\top = -4 \cdot 1(X) + 3a_n(X) + 2a_{n-1}(X)$, which gives the first element of Gröbner basis, namely $a_n^2 + 4 - 3a_n - 2a_{n-1}$. Similarly, we can express the remaining three relevant monomials in terms of the vector basis V_B to obtain the finite set $G = \{a_n^2 + 4 - 3a_n - 2a_{n-1}, a_n a_{n-1} + 3 - a_n - 3a_{n-1}, a_{n-1}^2 + 2 - 3a_{n-1}, a_{n-2} - a_n + a_{n-1}\}$. We observe that this is the same Gröbner basis as the one obtained in Section 4.1.1. In particular, $I_{123} = \langle G \rangle$.

Additional examples can be found in [111].

4.1.4 Discovery of exact equations via Gröbner bases

Having established the relation between the computation of Gröbner bases and the task of equation discovery, we can now introduce *MoadeeB*, a novel method for discovering exact equations from data. It stands for Möller–Buchberger algorithm–based discovery of exact equations. Presented in Algorithm 4.1, it requires data X in the form of a matrix and configurable parameters *len*, *mag*, and *top_n*, as described in the following text. It returns a list of *top_n* polynomial equations that hold for the data, sorted from the simplest to the most complex.

Algorithm 4.1: *MoadeeB* — Möller–Buchberger algorithm–based discovery of exact equations

Input: Observation matrix X of variables $V = \{x_1, x_2, \dots, x_p\}$.
Input: Parameters (*len*, *mag*, *top_n*) controlling equation complexity and quantity.
Output: Sorted list L of the *top_n* simplest polynomial equations of the form $f(x_1, x_2, \dots, x_p)$=0.

```
1 Points = Unique(X)
2 GröbnerBasis = MöllerBuchberger(Points)
3 L = Filter(GröbnerBasis, len, mag)
4 L = MagSort(L, top_n)
5 return L
```

The function *Unique* (line 1) ensures no data point is repeated, as the Möller–Buchberger algorithm does not allow duplicates. A Gröbner basis outputted by the Möller–Buchberger algorithm (line 2) is then filtered (line 3) and sorted (line 4) to extract the simplest *top_n* polynomials. We measure the complexity of a polynomial according to the number of

terms (*len*) and the magnitudes of the polynomial coefficients (*mag*). In particular, the magnitude equals $\sum_c N_n(c) + N_d(c)$, where the index c goes through the coefficients of the polynomial, while N_n and N_d are functions that return the number of the digits in the numerator and the denominator of c, respectively. When c is an integer, we assume that $N_d(c) = 0$. The function *Filter* (line 3) keeps only polynomials with a number of terms at most *len* and magnitude at most *mag*. The function *MagSort* (line 4) first sorts the polynomials based on their magnitude complexity and then sorts only the subset of polynomials with magnitude complexity of up to 20 by the number of terms. In the end, it returns the first *top_n* polynomials.

4.1.5 Time complexity of discovering exact equations via Gröbner bases

The bit complexity of the Möller–Buchberger algorithm [70], expressed in big-O notation, is $O(m^4 p(m \log^2 m + p))$ bit operations [70], [71], under certain asymptotic assumptions that bound the magnitude of the input values and using the bound $g \leq mp$ on the number of Gröbner basis elements, where m and p denote the number of data points (rows) and variables (columns) in the data set, respectively. In our specific case involving integer sequences, the number of columns corresponds to the selected recurrence order (denoted o), giving $p = o+1$. Additionally, since we use only the first $2 \cdot o+10$ terms of each sequence, the number of rows is $m = o+10$. Substituting these values, the time complexity simplifies to $O(o^5 \log^2 o + o^6)$, which is asymptotically $O(o^6)$. Notably, this represents a significant reduction in computational complexity compared with *Diofantos*, whose time complexity is polynomial with a degree depending on $d_{\max}$, or even exponential in this parameter, as analyzed in Subsection 3.1.5.

SINDy, on the other hand, has a theoretical time complexity of $O(p^{2d_{\max}} \cdot m)$, where m, p, and $d_{\max}$ denote the number of rows, columns, and the maximum polynomial degree, respectively. This complexity arises primarily from the least-squares algorithm employed by SINDy, which has a time complexity of $O(p^2 m)$. As SINDy generates additional columns corresponding to higher-degree monomials up to degree $d_{\max}$, the resulting time complexity becomes $O((p^{d_{\max}})^2 m)$. While we utilized SINDy's sequentially thresholded least squares algorithm, default settings limited it to a maximum of 20 iterations. Since this only multiplies the number of computational steps by a constant, it does not affect the asymptotic time complexity. This asymptotic bound also applies in the bit-complexity model, since SINDy uses fixed-precision floating-point arithmetic, for which each arithmetic operation can be treated as taking constant time.

In our specific application to integer sequences, the number of columns is determined by the recurrence order o, giving $p = o+1$, while the number of rows m satisfies $m = O(N)$. This results in a time complexity of $O(o^{2d_{\max}} N)$. Consequently, the time complexity of SINDy grows exponentially with the maximum polynomial degree considered. In comparison with *MoadeeB*, the asymptotic time complexity of SINDy eventually exceeds that of *MoadeeB* as $d_{\max}$ increases. For instance, when polynomials of degree up to three are considered, the dependence on o already matches the dominant $O(o^6)$ term of *MoadeeB*, and grows faster for larger values of $d_{\max}$.

4.2 Empirical Evaluation

In this chapter, we evaluate *MoadeeB* on the three OEIS benchmarks introduced in Section 1.6.2: *linrec*, *core*, and the TM-OEIS benchmark. These experiments follow the common evaluation framework established there: *linrec* provides a large-scale benchmark for reconstructing known linear recurrences, *core* provides a smaller and more heterogeneous

collection on which both reconstruction and the discovery of potentially new equations can be studied, and the TM-OEIS benchmark enables comparison under the predictive protocol of [34]. The main comparison is with *Diofantos* and SINDy [25], while QSynt [38], the computer-algebra Guessing approach implemented in `ore_algebra` [18], and TM-OEIS [34] provide additional sequence-specific reference points under their respective evaluation protocols. The following subsection specifies the compared algorithms, their settings, and the assessment criteria used in these experiments; the subsequent subsections present and discuss the corresponding results.

4.2.1 Experimental setup

Compared algorithms and settings. Our main experimental comparison is between *MoadeeB*, *Diofantos* (Chapter 3), and SINDy [25]. Additional comparisons with sequence-specific methods—QSynt [38], the computer-algebra Guessing approach implemented in `ore_algebra` [18], and TM-OEIS [34]—are reported in Subsection 4.2.4. As discussed in Section 1.6.2, these methods do not all produce the same type of output or support the same evaluation protocol, and their results are therefore treated as complementary reference points rather than as directly interchangeable baselines.

For *Diofantos* and SINDy, we retain the experimental settings described in Section 3.2.1. In particular, their procedures for selecting sequence terms and constructing the corresponding tabular data remain unchanged, including the floating-point-driven truncation applied to SINDy.

The *MoadeeB* protocol follows the same general progression over recurrence orders but is adapted to control the computational cost of Gröbner basis construction. At recurrence order o, *MoadeeB* uses the first $2o + 10$ terms of the sequence to construct the input data. Unlike *Diofantos*, *MoadeeB* does not impose a predefined maximal polynomial degree and does not perform a separate search over degree values. Instead, the candidate relations are obtained directly from the Gröbner basis computed at each recurrence order.

From each Gröbner basis, we select the ten simplest candidate equations according to the *mag* and *len* criteria defined in Section 4.1.4. These candidates are then validated against all sequence terms available in the corresponding benchmark. The search terminates as soon as at least one candidate is found to be valid on the complete available sequence prefix.

For the Möller–Buchberger step, we use the implementation of [70] with its default settings, including the `DegRevLex` monomial ordering. Full pseudocode for the *linrec* and *core* procedures is provided in Appendix B, Algorithms B.1 and B.2.

On the *linrec* benchmark, the search is additionally tailored to the benchmark's known hypothesis class. If an explicit linear recurrence is found, *MoadeeB* returns the first such recurrence; if no linear recurrence is discovered, the nonlinear candidates obtained during the search are retained instead. Giving preference to explicit linear recurrences makes the results more directly comparable with those of *Diofantos* and SINDy, whose searches on *linrec* are restricted to linear recurrence relations.

Performance assessment. As in the common framework of Section 1.6.2, we distinguish between exact validity on the observed sequence terms and reconstruction of a documented relation. A discovered equation is considered valid when it holds exactly for every applicable instance in the available sequence prefix. Reconstruction requires the stronger condition that the discovered relation be mathematically equivalent to the known reference relation.

For *linrec*, this chapter uses a stronger reconstruction criterion than the coefficient-vector comparison employed in Chapter 3. Equality of coefficient vectors recognizes only identical representations and may therefore fail to identify recurrences that are mathemati-

cally equivalent but written in different forms. Here, equivalence is instead determined automatically by comparing the rational generating functions associated with the discovered and documented linear recurrences. Two recurrences are counted as a reconstruction when they define the same generating function, and hence the same sequence. The construction and comparison of these generating functions are described in detail in Appendix C.

This equivalence test makes it possible to more reliably distinguish between failure to reproduce the documented coefficient vector and failure to recover the underlying recurrence relation. It therefore instantiates, for linear recurrences, the mathematically meaningful reconstruction criterion introduced in Section 1.6.2.

For the *core* benchmark, the evaluation follows the same principles as in the preceding chapter. Discovered equations are first required to hold exactly on all available sequence terms. Reconstruction of documented relations is assessed manually where applicable, owing to the heterogeneous forms of the equations associated with core sequences. Valid equations that do not correspond to documented relations are considered candidates for novelty and are analyzed according to the procedure outlined in Section 1.6.2: they are compared with existing OEIS information, and their general validity is established manually by mathematical proof where possible before previously undocumented relations are considered for submission to the OEIS curators.

The comparison with TM-OEIS follows the predictive protocol of [34], rather than the reconstruction criterion used for *linrec*. Similarly, comparisons with QSynt and Guessing use the evaluation procedures appropriate to their respective output representations and hypothesis classes. As emphasized in Section 1.6.2, these results therefore provide complementary evidence about OEIS equation discovery but are kept conceptually separate from reconstruction rates based on mathematical equivalence.

The remainder of this section presents the empirical results in the same order as these evaluation objectives. We first examine reconstruction of known linear recurrences on *linrec*, followed by exact equation discovery on the *core* sequences. We then compare *MoadeeB* with the sequence-specific reference approaches and analyze its relationship with *Diofantos*. Finally, we report the approximate total wall-clock time required to complete the experiments for each data set on the HPC cluster.

4.2.2 Linear recurrence reconstruction on OEIS

Table 4.2 summarizes the performance of *MoadeeB* on reconstructing linear recurrence relations from the OEIS. Out of 27,236 sequences with documented linear recurrences, *MoadeeB* exactly reconstructed the OEIS-reported recurrence for 10,222 sequences (37.5%). In addition, it discovered 315 recurrences (1.2%) that are not syntactically identical but are provably equivalent to the OEIS ground truth. Equivalence between linear recurrences was established using the procedure described in Appendix C.

Beyond these provable reconstructions, *MoadeeB* discovered valid equations for a substantial number of additional sequences. Specifically, for 14,613 sequences, *MoadeeB* produced non-linear polynomial equations that are correct on all available sequence elements but cannot be proved equivalent to the OEIS-reported linear recurrences, as the equivalence test in Appendix C applies only to linear recurrences. While these equations are more complex than the documented linear recurrences and therefore of limited practical interest in this setting, they nonetheless represent valid exact relations. For 2,086 sequences (7.7%), *MoadeeB* failed to identify any equation consistent with the data. Overall, *MoadeeB* discovered at least one valid equation for 92.3% of the linear-recurrent OEIS sequences, while its provable reconstruction rate (ID+EQUIV) is 38.7%.

When considering the ability to discover any valid equation, *MoadeeB* achieves a lower failure rate than the three comparison methods, whose failure rates range from 9.1% (*Dio-*

Table 4.2: Performance comparison of *MoadeeB*, *Diofantos*, and SINDy on the *linrec* data set. For each method, ID denotes sequences for which the discovered equation is identical to the OEIS-reported recurrence, EQUIV denotes equations provably equivalent to the OEIS-reported recurrence, and VALID denotes equations that are correct on all available sequence elements but whose equivalence could not be established. FAIL denotes sequences for which no valid equation was found, while ID+EQUIV reports the number of provably reconstructed OEIS recurrences.

		Different			
Approach	**ID**	**EQUIV**	**VALID**	**FAIL**	**ID+EQUIV**
MoadeeB	**10222**	315	14613	**2086**	10537
Diofantos	9206	**15210**	345	2475	**24416**
SINDy-default	5678	11821	1	9736	17499
SINDy-tuned	6445	12592	3	8196	19037

fantos) to 35.7% (SINDy with default settings). However, when evaluation is restricted to provable reconstruction of the OEIS-reported linear recurrences, the other methods achieve substantially higher reconstruction rates, ranging from 64.2% (SINDy-default) to 89.6% (*Diofantos*). This discrepancy is primarily due to differences in inductive bias: both *Diofantos* and SINDy are explicitly constrained to search for linear equations in this experiment, whereas *MoadeeB* is capable of discovering non-linear relations and is not restricted to the linear hypothesis class.

Table 4.3: Breakdown of equations classified as VALID in Table 4.2. The table distinguishes between equations that are provably not equivalent to the OEIS-reported linear recurrence and equations whose equivalence could not be verified using the linear-recurrence equivalence test described in Appendix C.

Approach	**Proved Non-equivalent**	**Non-verifiable**
MoadeeB	70	14543
Diofantos	57	288
SINDy-default	**0**	1
SINDy-tuned	**0**	3

Table 4.3 provides a more detailed breakdown of cases in which equivalence to the OEIS ground truth could not be established. For *Diofantos* and SINDy, the dominant cause is insufficient sequence length to perform the equivalence test; in addition, for 57 sequences, *Diofantos* produced equations that failed to generalize beyond the training data. In contrast, the reasons differ for *MoadeeB*. For only 70 sequences, the discovered equations failed to reconstruct sequence elements outside the training set. For the remaining 14,543 sequences with valid but non-equivalent solutions, equivalence could not be verified because *MoadeeB* discovered implicit polynomial equations involving non-linear terms, for which automated equivalence testing against linear recurrences is not applicable.

4.2.3 Exact equation discovery on core OEIS

Table 4.4 reports the results of exact equation discovery on the core OEIS sequences. On this benchmark, *MoadeeB* reconstructs nine more equations than *Diofantos* and additionally discovers two previously unknown equations. Both novel equations were rigorously

verified and subsequently reviewed and accepted by the OEIS curators. This represents a substantial improvement over prior results on this data set, where *Diofantos* and SINDy exhibited comparable performance, differing by only two successful reconstructions under SINDy's default settings.

Table 4.4: Performance comparison of *MoadeeB*, *Diofantos*, and SINDy on the core OEIS sequences, grouped by sequence category. The table reports the number of successful reconstructions obtained by each method.

Approach	**Trivial (4)**	**Simple (66)**	**Other (94)**	**All (164)**
MoadeeB	4	**43**	0	**47**
Diofantos	4	32	0	36
SINDy-default	4	30	0	34
SINDy-tuned	4	31	0	35

The improvement on the core sequences is particularly significant, as these sequences are identified by OEIS curators as mathematically central and are often the subject of ongoing research. Table 4.5 summarizes all equations reconstructed by *MoadeeB*for which *Diofantos*and SINDy failed. It also highlights the two newly discovered equations.

Closer inspection of cases where only *MoadeeB* succeeds reveals a common structural pattern: all additional reconstructions involve recurrence relations with integer division. For example, *MoadeeB* reconstructs the recurrence $a_n = \frac{2(2n+1)a_{n-1}}{n+1}$ (sequence A001700) in its implicit polynomial form $na_n - 4na_{n-1} + a_n - 2a_{n-1} = 0$. Similarly, *MoadeeB* reconstructed $a_n = \frac{(2n-1)a_{n-1}+3(n-1)a_{n-2}}{n}$ (sequence A002426) as $na_n - 2na_{n-1} - 3na_{n-2} + a_{n-1} + 3a_{n-2} = 0$. Such equations are inaccessible to *Diofantos*, which operates strictly over integer-valued relations, and to SINDy, whose standard formulations rely on floating-point regression and coefficient rounding. In principle, extending *Diofantos* with integer-kernel solvers could enable the discovery of some implicit equations involving rational coefficients. Similarly, SINDy's limitations could be mitigated by adopting variants designed for implicit equation discovery [78].

Analysis of the remaining core sequences suggests that a small number of additional equations could, in principle, be reconstructed by *MoadeeB*. These equations are all of degree three, whereas among the successfully reconstructed equations, only two have degree three, and none have a higher degree. This behavior is consistent with the increasing difficulty of discovering higher-degree polynomial relations.

Finally, we note that not all implicit equations provide meaningful insight into the underlying structure of a sequence. For instance, for the Kolakoski sequence (A000002), *MoadeeB* discovers the equation $a_n^2 - 3a_n + 2 = 0$, which merely encodes the fact that $a_n \in \{1, 2\}$, and thus adds no new information. A similar phenomenon occurs for OEIS A000120 (the number of ones in the binary expansion of n), where *MoadeeB* produces $a_n a_{n-1} - a_{n-1}^2 - a_n a_{n-2} + a_{n-1}a_{n-2} - a_n + a_{n-2} + 1 = 0$, which factorizes as $(a_n - a_{n-1} - 1)(a_{n-1} - a_{n-2} - 1) = 0$. This relation reflects a simple property of the sequence—namely, that among any three consecutive terms, at least one adjacent pair differs by one—and illustrates that exact discovery does not always imply conceptual novelty.

4.2.4 Comparison with state-of-the-art OEIS discovery methods

This subsection presents the additional comparisons with QSynt, Guessing, and TM-OEIS listed in Table 1.2. For QSynt and TM-OEIS, we extend the comparisons described in

Table 4.5: Equations reconstructed or discovered by *MoadeeB* for core OEIS sequences on which both *Diofantos* and SINDy failed. The table reports the OEIS identifier, sequence name, and equation discovered by *MoadeeB*. The final two entries correspond to previously undocumented equations that were subsequently proved rigorously and accepted into the OEIS.

ID	Short Name	Discovered Equation
A000108	Catalan nos.	$a_n = 2(2n-1)a_{n-1}/(n+1)$
A000984	$\binom{2n}{n}$	$n \cdot a_n + 2(1-2n)a_{n-1} = 0$
A001003	Lit. Schröder no.	$(n+1)a_n = (6n-3)a_{n-1} - (n-2)a_{n-2}$
A001006	Motzkin nos.	$(n+2)a_n = (2n+1)a_{n-1} + (3n-3)a_{n-2}$
A001405	$\binom{n}{\lfloor n/2 \rfloor}$	$(n+1)a_n = 2a_{n-1} + 4(n-1)a_{n-2}$
A001700	$\binom{2n+1}{n+1}$	$a_n = 2(2n+1)a_{n-1}/(n+1)$
A002426	Centr. trin. coef.	$a_n = ((2n-1)a_{n-1} + 3(n-1)a_{n-2})/n$
A006318	Lar. Schröder no.	$(n-2)a_{n-2} - 3(2n-1)a_{n-1} + (n+1)a_n = 0$
A006894	Planted 3-trees	$a_n = a_{n-1}(a_{n-1}+1)/2 + 1$
A001699	Binary trees	$a_n = -a_{n-2}^3 + a_{n-1}^2 + 3a_{n-1}a_{n-2} + 2a_{n-2}^2 + 2a_{n-1} - 4a_{n-2}$
A002658	Planted 3-trees	$a_n = -\frac{a_{n-2}^3 - 2a_{n-1}^2 - 6a_{n-1}a_{n-2} - 4a_{n-2}^2 - 8a_{n-1} + 11a_{n-2}}{4}$

Subsection 3.2.4 to *MoadeeB*. The Guessing experiments are restricted to the *core* benchmark.

For QSynt, we use the results reported in [38]. Following the comparison procedure in Subsection 3.2.4, we validate the equations discovered by *MoadeeB* against the available sequence prefixes of up to 200 terms in our benchmarks.

Table 4.6: Performance comparison of *MoadeeB*, *Diofantos*, Guessing, and QSynt on the *linrec* and *core* data sets. Results are measured by the number of reconstructed equations or valid programs. The entry All† indicates that Guessing is theoretically guaranteed to reconstruct all linear recurrences. The column labeled Other reports QSynt-discovered programs for OEIS sequences not included in either the *linrec* or *core* data set. The totals reported in the QSynt paper do not equal the sum of the *linrec*, *core*, and Other columns because 30 core sequences also appear in the *linrec* data set. Italics indicates the result obtained by the proposed method on the core OEIS sequences.

Approach	*linrec*	*core*	Other
MoadeeB	25,150	*47*	-
Diofantos	24,761	36	-
Guessing	**All**†	44	-
QSynt	14,168	**50**	13,798

†The method is theoretically guaranteed to be complete for linear recurrences under certain assumptions; however, these assumptions may not hold for the *linrec* dataset due to the limited number of training elements in each sequence.

The comparative results are summarized in Table 4.6. On the linear-recurrence benchmark, *MoadeeB* reconstructs more valid equations than QSynt. On the core OEIS sequences, QSynt discovers a larger number of valid programs than *MoadeeB* finds valid equations. This result narrows the previously reported gap between symbolic regression and pro-

gram synthesis, but also reflects the greater expressive power of QSynt's Turing-complete hypothesis space. Sequences such as A000169 ($a_n = n^{(n-1)}$), A000272 ($a_n = n^{(n-2)}$), and A000312 ($a_n = n^n$) illustrate cases where program synthesis naturally captures growth patterns that are difficult to express compactly using algebraic equations.

For the Guessing methods reviewed in Section 2.6, we restrict the experiments to the *core* benchmark. We omit *linrec*, which consists exclusively of sequences with constant-coefficient linear recurrences, for which the Berlekamp–Massey algorithm provides recovery guarantees given sufficiently many sequence terms. All Guessing experiments are conducted using the `guess` function of the open-source Sage `ore_algebra` package [18], [112] with default parameters.

The results, reported in Table 4.6, show that *MoadeeB* reconstructs three more equations than Guessing on the core benchmark. In particular, *MoadeeB* identifies four equations missed by Guessing (OEIS A000058, A001699, A002658, and A006894), two of which correspond to the original discoveries described in Section 4.3. All four equations involve non-linear interactions between past sequence elements, placing them outside the expressive scope of Guessing methods, which assume relations of the form $\sum_{i=0}^{k} p_i(n) \cdot a_{n-i} = 0$ with polynomial coefficients $p_i(n)$.

Conversely, Guessing recovers a linear recurrence with polynomial coefficients missed by *MoadeeB*, $2n(2n+1)a_n - 3(3n-1)(3n-2)a_{n-1} = 0$, whose high polynomial degree poses a challenge for *MoadeeB*. This highlights a complementary trade-off: Guessing excels at high-degree linear recurrences, whereas *MoadeeB* can recover non-linear relations that are inaccessible to linear-recurrence-based approaches.

We compare *MoadeeB* with the recurrence-inference setting of TM-OEIS [34], using the 10,000-sequence benchmark and predictive evaluation protocol described in Subsection 3.2.4. For this comparison, *MoadeeB* uses the search settings of the *core* experiments (Algorithm B.2), with the input prefixes specified by the TM-OEIS protocol. Since evaluation is based on predictive accuracy, we rewrite the discovered implicit equations into explicit form (line 8 of Algorithm B.2) to enable forward prediction.

Table 4.7: Performance comparison of *MoadeeB*, *Diofantos*, and TM-OEIS on 10,000 OEIS sequences [34]. Performance is evaluated by the predictive accuracy of the discovered equations. The four experimental settings are defined by the number of input sequence elements used for discovery, either 15 or 25, and the number of subsequent elements predicted, either 1 or 10. Boldface denotes the best-performing method for each experimental scenario.

Approach	$n_{input} = 15$		$n_{input} = 25$	
	$n_{pred} = 1$	$n_{pred} = 10$	$n_{pred} = 1$	$n_{pred} = 10$
MoadeeB	31.5	22.3	**35.0**	**26.8**
Diofantos	29.0	**22.6**	28.5	26.2
TM-OEIS	**33.4**	19.2	34.5	21.3

The results are shown in Table 4.7. When trained on 25 sequence elements, *MoadeeB* predicts the next 10 elements with an accuracy of 26.8%, outperforming all other methods and exceeding *Diofantos* by 54 correctly predicted sequences (out of 10,000). For predicting the next single element, *MoadeeB* achieves an accuracy of 35.0%, slightly surpassing TM-OEIS. With only 15 input elements, *MoadeeB* attains the second-best performance, predicting the next 10 elements with 22.3% accuracy (33 fewer correct predictions than *Diofantos*) and the next single element with 31.5% accuracy, which is approximately 2% below TM-OEIS.

4.2.5 Relation between *MoadeeB* and *Diofantos*

The experimental results highlight two key ways in which *MoadeeB* generalizes *Diofantos*, both of which have practical consequences for exact equation discovery.

First, *MoadeeB* naturally supports discovery from rational data, whereas *Diofantos* is restricted to integer-valued data and integer coefficients. *MoadeeB* builds on the Möller–Buchberger algorithm, which takes as input a set of rational points and computes a Gröbner basis with rational coefficients using exact arithmetic. As a result, *MoadeeB* can discover exact equations involving rational coefficients directly, without requiring coefficient rounding or post-processing. This capability is empirically demonstrated in the last two data sets reported in Table 5.1, where *Diofantos* is not applicable.

Second, *MoadeeB* can discover equations in implicit form, while *Diofantos* is limited to explicit equations. This distinction is reflected in the core OEIS results (Table 4.5), where *MoadeeB* successfully reconstructs nine equations that *Diofantos* fails to recover. In all these cases, the recovered relations are implicit polynomial equations that cannot be expressed in the explicit form required by *Diofantos*.

Importantly, these two generalizations are complementary. Any polynomial equation with rational coefficients can be transformed into an implicit equation with integer coefficients by multiplying through by the denominators. However, since *Diofantos* cannot discover implicit equations, such transformed relations remain outside its hypothesis class. In contrast, *MoadeeB* handles both rational coefficients and implicit representations natively, which explains the observed empirical differences between the two methods.

4.2.6 Running times

Table 4.8: Running-time comparison of *MoadeeB*, *Diofantos*, and SINDy on the *linrec* and *core* data sets. Times are reported in approximate wall-clock units using days, hours (h), minutes (min), or seconds (s), as appropriate. A denotes the total time required for all experiments, and R denotes the time required for successful reconstructions.

Approach	***linrec* [R/A]**	***core* [R/A]**
MoadeeB	4 h	2 min / **2 min**
Diofantos	33 h / 2 days	10 h / 2 days
SINDy-default	**2 h**	**8 s** / 2 days
SINDy-tuned	6 h	1 h / 2 days

Table 4.8 presents the total approximate real-time required to complete experiments for all sequences in each data set, run on an HPC cluster. As in Table 3.8, the experiments were run in parallel, so the reported wall-clock times approximately reflect the maximum runtime required to process an individual sequence. *MoadeeB* exhibits significantly improved time complexity. Specifically, it is roughly 10 times faster than *Diofantos* on both data sets. Notably, *MoadeeB*'s runtime falls between the default and fine-tuned SINDy configurations. This suggests that, in this instance, fine-tuning SINDy incurs a substantial time penalty, exceeding *MoadeeB*'s runtime while achieving lower discovery success. On the *core* data set, *MoadeeB* further demonstrates its efficiency, being the only method to complete execution within the two-day time limit.

The primary explanation for *MoadeeB*'s reduced runtime is consistent with the theoretical complexity estimates discussed in Sections 4.1.5 and 3.1.5. Another factor contributing to *MoadeeB*'s reduced runtime is its use of only the first $2 \cdot o + 10$ sequence elements (e.g.,

line 3 of Algorithm B.1), compared with the other methods, which use all 200 elements. This reinforces *MoadeeB*'s robustness, which is also evident in the high success rates on the *linrec* and *core* data sets (Tables 4.2 and 4.4). Finally, particularly in comparison with SINDy, the reduced runtime may also be partly attributed to the faster implementation of the VALIDATE function used in the *MoadeeB* experiments. For *MoadeeB*, this function was implemented using the underlying CoCoA library [113] and without recursive function calls.

Overall, considering both discovery success and runtime performance, *MoadeeB* offers significant improvements over competing methods without sacrificing computational efficiency. However, as stated in Subsection 3.2.5, we report running times only to provide the reader with a general indication of the practical computational cost of our implementations. Running time alone is not an appropriate measure of the theoretical time complexity of an equation discovery algorithm.

4.3 Discovery of Novel OEIS Recurrences

Applying *MoadeeB* to the core OEIS benchmark led to the discovery of two previously undocumented recurrence relations, detailed in Subsection 4.2.3. For the sequence counting binary trees by height (A001699), we identified the recurrence

$$a_n = -a_{n-2}^3 + a_{n-1}^2 + 3a_{n-1}a_{n-2} + 2a_{n-2}^2 + 2a_{n-1} - 4a_{n-2}.$$

Similarly, for the sequence of planted 3-tree counts (OEIS A002658), we discovered the relation

$$a_n = -\frac{a_{n-2}^3 - 2a_{n-1}^2 - 6a_{n-1}a_{n-2} - 4a_{n-2}^2 - 8a_{n-1} + 11a_{n-2}}{4}.$$

Both equations were subsequently rigorously proved and formally accepted into the corresponding OEIS entries, A001699 and A002658 [114], [115].

Chapter 5

Discovery From Non-OEIS Data

In the previous chapters, we evaluated methods based on Diophantine equation solving and Gröbner basis computation on data sets of integer sequences, focusing on the discovery of recurrences satisfied by those sequences. However, this evaluation still leaves two important questions open. First, it is natural to ask whether the developed methods are useful beyond the specific setting of the OEIS. Second, although our large-scale experiments rigorously studied linear equations, the broader applicability of the approach to more general classes of equations was not examined at a comparable scale. In particular, while one data set used in the comparison with neural sequence modeling approaches comprised 10,000 sequences, that analysis evaluated only next-term predictive accuracy, rather than exact recovery of equations equivalent to the ground truth.

In the remainder of this chapter, we address both of these questions. In the first section, we consider a small but illustrative benchmark consisting of 14 equations, including 10 well-known examples from the mathematics literature, designed to demonstrate the utility of the proposed methods outside the OEIS setting. In the second section, we introduce a large-scale synthetic benchmark comprising 3,020 equations over tabular data generated from implicit, rational, and polynomial relations. Together, these benchmarks provide a broader and more demanding evaluation of the proposed methods.

5.1 Utility of Exact Equation Discovery Outside OEIS

We evaluated *Diofantos* and *MoadeeB* on a set of fourteen real-world benchmarks to assess their generalizability beyond the OEIS and the domain of recursive equations for integer sequences. We selected benchmarks from well-known equations in basic mathematics, linear algebra, graph theory, and algebraic geometry, as well as from our own selection of simple symbolic computation tasks. Table 5.1 presents 14 selected exact equations across 9 data sets.

We generated each of the artificial data sets in the following manner. For the first equation, we copied 50 Pythagorean triplets from the corresponding Wikipedia page. For the second and third data sets, we generated 100 random square matrices A and B of dimensions 2×2 and 3×3. For the fourth data set, we selected the first 100 wheel graphs W_n, with each graph corresponding to the number of nodes n it contains. We generated the fifth data set by generating 100 random planar graphs with a random number of components $|\Omega|$ ranging from 1 to 10, and a random number of nodes in each component ranging from 3 to 100. For the sixth data set, we first took 100 pairs of random integers between 0 and 100 corresponding to the degree $\deg(D)$ of a randomly chosen divisor D (with non-negative degree) on a random compact Riemann surface of genus g, while keeping $\deg(D)$ bigger than g. Then we calculated the corresponding dimension $\ell(D)$ via the

Table 5.1: Overview of the non-OEIS mathematical benchmark equations. The fourteen benchmark equations are grouped by data set identifier. For each entry, the table reports the equation name, governing equation, and variables used for equation discovery.

Data Set	**Name**	**Equation**	**Variables**
1	Pythagorean eq.	$a^2 + b^2 = c^2$	a, b, c
2	add. of det. homo. of det.	$\det(A) \cdot \det(B) = \det(AB)$ $\det(\alpha \cdot A) = \alpha^2 \cdot \det(A)$	$\det(A), \det(B)$, $\det(AB), \alpha$, $\det(\alpha \cdot A)$
3	add. of trace com. of tr.	$\mathrm{tr}(A + B) = \mathrm{tr}(A) + \mathrm{tr}(B)$ $\mathrm{tr}(AB) = \mathrm{tr}(BA)$	$\mathrm{tr}(A)$, $\mathrm{tr}(B)$, $\mathrm{tr}(A + B)$, $\mathrm{tr}(AB)$, $\mathrm{tr}(BA)$
4	edg. of wheel wheel's min. deg. wheel's max. deg.	$\lvert E(W_n)\rvert = 2n$ $\delta(W_n) = 3$ $\Delta(W_n) = n$	$n, \lvert V(W_n)\rvert$, $\lvert E(W_n)\rvert$, $\delta(W_n), \Delta(W_n)$
5	Euler's formula	$\lvert V\rvert - \lvert E\rvert + \lvert F\rvert = 1 + \lvert\Omega\rvert$	$\lvert V\rvert, \lvert E\rvert, \lvert F\rvert, \lvert\Omega\rvert$
6	Riemann-Roch	$\ell(D) = \deg(D) - g + 1$	$\ell(D), \deg(D), g$
7	symb. comp. 1 symb. comp. 2	$w_1 := 1 + x + 3y + 2xy - x^2 - y^2$ $= 2\alpha - \beta^2 - \beta + 1,$ where $\alpha = x + y, \beta = x - y$ $w_2 := 2 + x - y - 4xy -$ $-2x^3 + 6xy^2$ $= \alpha^3 - \alpha^2 + \beta^3 + \beta^2 + \beta + 2$	α, β, w_1, w_2
8	symb. comp. 3	$w_3 := \frac{1}{y}(y^2 - x^2) = \frac{1}{y}\beta^2 - \frac{x}{y} \cdot 2\beta$	$w_3, \frac{1}{y}, \frac{x}{y}, \frac{1}{y}\alpha, \frac{x}{y}\alpha,$ $\frac{1}{y}\beta, \frac{x}{y}\beta, \frac{1}{y}\alpha^2, \frac{x}{y}\alpha^2,$ $\frac{1}{y}\beta^2, \frac{x}{y}\beta^2, \frac{1}{y}\beta^3$
9	symb. comp. 4	$w_3 := \frac{1}{y}(-3x^2 + 3y^2 + 3y)$ $= \frac{1}{y}(\frac{3}{2}\alpha + 3\beta^2 - \frac{3}{2}\beta) - \frac{x}{y} \cdot 6\beta$	

formula to introduce the corresponding column of the data set.

For these six data sets, both *Diofantos* and *MoadeeB* successfully reconstructed the ten corresponding equations listed in Table 5.1. The only exception was the first case, the Pythagorean equation, for which we created an additional target column containing the squared variable c^2. This modification was necessary to avoid the implicit form of the equation, which *Diofantos* is, by default, unable to handle.

The last three data sets, involving four equations, represent symbolic computation tasks of rewriting an expression involving x and y into a specific form known in advance. The form does not, however, reveal the details of the rewritten equation. The first two equations (the seventh data set) require the form $f(\alpha) + g(\beta)$, while the last two equations require the form $\frac{1}{y}(f(\alpha) + g(\beta)) - \frac{x}{y}(f'(\alpha) + g'(\beta))$, where $\alpha := x + y$ and $\beta := x - y$. Our approach to solving this task involves generating a data set with five columns corresponding to the variables α, β, $\frac{1}{y}$, $\frac{x}{y}$, and $w(x, y)$ (specifically w_1, w_2, or w_3 in the table). One can randomly choose multiple values of x and y, and then calculate α, β, $\frac{1}{y}$, $\frac{x}{y}$, and $w(x, y)$ based on these values, where $w(x, y)$ is the starting expression involving x and y. Applying *Diofantos* and *MoadeeB* to the resulting data set may reveal the right-hand side of the equation in the required form. In our computer experiments, we included columns α and β, as well as both columns w_1 and w_2 in the seventh data set and reconstructed

both corresponding equations. For this data set, we generated 100 random integer values of x and y between -10 and 10. On the other hand, the reconstruction of the eighth and ninth equations did not proceed as smoothly, so we introduced additional columns to assist *MoadeeB*, resulting in successful reconstruction. Having only columns w_3, $\frac{1}{y}$, $\frac{x}{y}$, α, and β in the data sets resulted in equations with mixed monomials containing α and β simultaneously, which is not compatible with the required form. However, after including columns corresponding to values of the variables w_3, $\frac{1}{y}$, $\frac{x}{y}$, $\frac{1}{y}\alpha$, $\frac{x}{y}\alpha$, $\frac{1}{y}\beta$, $\frac{x}{y}\beta$, $\frac{1}{y}\alpha^2$, $\frac{x}{y}\alpha^2$, $\frac{1}{y}\beta^2$, $\frac{x}{y}\beta^2$, and $\frac{1}{y}\beta^3$, *MoadeeB* successfully reconstructed the desired linear combinations. This validates the well-known general rule that incorporating background knowledge during data set creation can significantly enhance the performance of equation discovery algorithms by removing certain columns or creating new ones. For these two data sets, we generated 100 random integer values of x and y between -30 and 30, ensuring that y remained nonzero to avoid division by zero. Since the division of random integers can quickly result in rational numbers, these two data sets also demonstrate *MoadeeB*'s capability of handling rational numbers, in contrast to *Diofantos*, which only deals with integers. For this reason, *Diofantos* was excluded from the experiments involving these two data sets. The specific symbolic computation tasks were chosen to illustrate the potential utility of *MoadeeB*, *Diofantos*, and other methods to obtain more comprehensive solutions of partial differential equations from special ones. A good introduction to these methods, known as variation of parameters, is given by [116], including numerous references to earlier work dating back to Lagrange.

Using the same experimental setup as for the *core* OEIS data set, *Diofantos* and *MoadeeB* successfully reconstructed nearly all equations listed in Table 5.1. Only the last two equations were reconstructed exclusively by *MoadeeB*, owing to the involvement of rational numbers. The result illustrates the ability of presented approaches to discover exact mathematical equations beyond OEIS. All nine data sets presented in the table are openly available at our repository [117].

5.2 Synthetic Non-OEIS Benchmark for Exact Equation Discovery

In the final round of computational experiments, we evaluate the proposed methods on the synthetic non-OEIS benchmarks summarized in Table 1.2, following the common evaluation framework introduced in Section 1.6.2. The polynomial and rational benchmarks are used to compare *Diofantos*, *MoadeeB*, and SINDy, whereas the implicit benchmark is evaluated only with *MoadeeB*.

The equations and corresponding data sets are publicly available in our repository [118], in the directory `evalu/EEDBench/`, specifically in the files `polys_map.json`, `ratios_map.json`, and `implicits_map.json`, as well as in the directories `polys`, `ratios`, and `implicits`.

5.2.1 Probabilistic context-free grammars

To generate the synthetic benchmarks, we used probabilistic context-free grammars [27], [119]. A probabilistic context-free grammar consists of rewrite rules equipped with probabilities and therefore defines a probability distribution over valid expressions. It can also be viewed as a random generative process that starts from a root symbol and recursively applies rewrite rules until the abstract syntax tree of an expression is fully generated. In our case, the grammars were used to generate mathematical expression structures, that is, expressions whose constants remain uninstantiated.

For polynomial equations, we used the following grammar:

$$\begin{array}{lrl} P \to & P + M\ [0.4] & \mid M\ [0.6] \\ M \to & M * V\ [0.4] & \mid c\ [0.6] \\ V \to & x\ [0.2] & \mid y\ [0.2] \quad \mid z\ [0.2] \quad \mid w\ [0.2] \quad \mid v\ [0.2] \end{array}$$

For rational equations, we used:

$$\begin{array}{lrl} R \to & (P)/(P)[0.2] & \mid P\ [0.8] \\ P \to & P + M\ [0.4] & \mid M\ [0.6] \\ M \to & M * V\ [0.4] & \mid c\ [0.6] \\ V \to & x\ [0.2] & \mid y\ [0.2] \quad \mid z\ [0.2] \quad \mid w\ [0.2] \quad \mid v\ [0.2] \end{array}$$

Here, P denotes a polynomial, M a monomial, V a variable, and c a constant. The probabilities shown in brackets correspond to the probabilities of applying the individual rewrite rules during generation.

The two grammars differ only in the additional start symbol R and its associated production rules in the rational case. These rules allow the generator to produce either a rational expression of the form $(P)/(P)$ or, with probability 0.8, a polynomial expression P. As a result, the rational grammar can also generate polynomial equations.

For the polynomial and implicit benchmarks, we used the polynomial grammar. The rational grammar was used only for the rational benchmark.

5.2.2 Generating equations and data sets

The probabilistic grammars introduced in the previous subsection generate expression structures rather than fully specified equations. For example, a generated structure may take the form $\frac{1}{x^2+c\cdot zy}$, where c denotes a placeholder constant to be instantiated later.

After generating a set of expression structures, we simplify them and retain only those that are both simplified and structurally unique. Each remaining structure is then turned into a concrete expression by assigning random integer values to all constants from the interval $[-10, 10]$. The resulting expressions are subsequently checked for symbolic equivalence, and only non-equivalent expressions are retained.

Each final expression is then used to construct a tabular data set. To this end, we first generate a table with 5 input columns, corresponding to the five independent variables, and 20 rows. The entries are sampled uniformly at random from the integer interval $[-10, 10]$. The expression is then evaluated on each row, and the resulting values are appended as the target column.

Of the 1000 expressions retained in the rational benchmark, 520 were generated through the direct production rule $R \to P$ and were therefore polynomial before constant instantiation.

Second, even when the grammar generates an expression of the form $(P)/(P)$, the denominator may simplify to a constant already at the structural level. For example, the denominator polynomial may be rewritten to a single constant term, which turns the overall expression into a polynomial with possibly rational-valued coefficients. Third, a structurally non-constant denominator may become constant only after instantiation of the placeholder constants, when the assigned values cancel all variable-dependent terms.

As a result, the final benchmark contains 791 polynomial and 209 genuinely rational equations. Many of the polynomial equations obtained in this way involve rational-valued coefficients, which in turn lead to non-integer values in the corresponding data sets. In our case, only 545 generated data sets have integer-valued entries only.

This structure of the benchmark is important when interpreting the results. *MoadeeB* is the only method considered here that is, in principle, suitable for the full task, since it

can recover rational equations, handle rational-valued coefficients, and operate on rational-valued data. Nevertheless, *Diofantos* and SINDy may still achieve a non-negligible success rate on this benchmark, precisely because a substantial portion of the generated instances are polynomial.

Generating data sets for implicit equations. Because *MoadeeB* is designed to discover implicit algebraic relations, we also constructed a benchmark of implicit polynomial equations in order to evaluate this capability explicitly. In contrast to explicit equations, an implicit equation of the form $f(\mathbf{x}) = 0$ does not directly determine one variable as a function of the others. Consequently, benchmark generation is less straightforward than in the explicit case: evaluating the left-hand side on randomly sampled data defines a new explicit target relation $y = f(\mathbf{x})$, rather than producing rows that satisfy the original implicit equation.

Our generation procedure starts from the same polynomial grammar as in the polynomial benchmark. In this case, however, we restrict attention to three variables in order to simplify the process of avoiding redundant equations. We generate unique polynomial expression structures, instantiate their constants, and retain only those equations for which a suitable data set can be constructed. The rational grammar was not used here, since rational implicit equations can be transformed into polynomial implicit equations by clearing denominators. For each candidate equation, we generate rows of the corresponding data set by first assigning random values to all variables except one, and then solving the resulting univariate polynomial equation for the remaining variable. Whenever rational solutions are obtained, they are used to form rows of the data set; otherwise, we continue by changing the selected variable or resampling the remaining values. If no suitable data set of 20 rows can be generated after repeated attempts, the equation is discarded.

To avoid uninformative or redundant benchmark instances, we excluded several classes of equations. First, univariate equations were discarded, since they do not encode any relation among variables, and the corresponding reconstruction task is essentially trivial for finite noise-free data. Second, we excluded equations that can be easily transformed into explicit form, because such cases are already covered by the explicit polynomial benchmark; the goal here was to evaluate genuinely implicit equations. Third, we attempted to reduce degenerate cases by requiring, with at most one exception, that each polynomial contains a nonzero constant term. Without this restriction, equations such as $x^2y^4 = 0$ are trivially satisfied whenever one of the variables is zero, producing rows that are equally compatible with a much simpler relation such as $xy = 0$.

Using this procedure, we obtained 20 genuinely implicit polynomial equations together with their corresponding data sets.

Limitations of the generated data sets for implicit equations. On the implicit benchmark, one source of false negatives arises when the method recovers an equation that is simpler than the generated ground truth but defines the same solution set. For example, for `ids00.csv`, *MoadeeB* discovers $xy = 0$ instead of $-7x^2y^2 = 0$. Although the recovered relation is algebraically simpler, it is symbolically different from the ground truth and was therefore automatically classified as a mismatch by our equivalence checker. However, over the real numbers, both equations define the same set of solutions and thus express essentially the same constraint. Consistent with the reconstruction criterion introduced in Section 1.6.2, we count this instance as a successful reconstruction because the two equations define the same solution set, despite the mismatch reported by the automatic symbolic check.

A different type of failure occurs when *MoadeeB* recovers only simple equalities that hold on the sampled points but do not capture the intended governing relation. This behavior appears in several instances, including `ids15.csv`, `ids16.csv`, `ids18.csv`, and

`ids19.csv`. For example, on `ids19.csv`, the method recovers $y = 1$ and $z^2 = 1$, indicating only that the y column is constant and that $z \in \{-1, 1\}$. Although both constraints are valid for the observed data, they are independent of x and therefore fail to capture the dependence structure encoded in the ground-truth equation. We regard such outcomes as unsuccessful reconstructions, since they reflect spurious invariants induced by limited sampling rather than genuine recovery of the underlying equation.

More generally, these cases suggest that the quality of the generated data sets for implicit equations may not always match that of the explicit benchmarks. In particular, some sampled rows may reflect only a restricted part of the underlying solution set, making the data appear more consistent with simpler relations than with the intended ground-truth equation. This points to a limitation of the current benchmark-generation procedure rather than of the reconstruction method alone.

5.2.3 Experimental setup

The experimental setup for the polynomial benchmark follows the applicable settings for *core* benchmark described in Subsections 3.2.1 and 4.2.1, with modifications for directly supplied tabular data. No sequence-to-table transformation or recurrence-order search is required. The initial experiments consider maximal polynomial degrees up to 3 for *Diofantos* and SINDy; *MoadeeB* does not use a maximal-degree parameter.

For *MoadeeB*, all 20 observations are used to compute the Gröbner basis, so no separate internal check of validity on the observed data is performed. For SINDy, we evaluate only the default configuration and use a faster implementation of the VALIDATE function to address the computational-budget constraints encountered in the OEIS experiments. Here, reconstruction is assessed using the symbolic equivalence check described in Section 1.6.2, implemented with SymPy [120].

We also evaluated alternative degree-selection strategies and parameter configurations motivated by initial exploratory runs and implementation checks. These revealed that the *Diofantos* method is sensitive to the degree-selection strategy. In particular, directly allowing polynomial degree 3 (denoted by *Diofantos (no loop)* in Table 5.2) often yielded better results than the incremental degree-selection strategy (denoted by *Diofantos* in Table 5.2) used in the OEIS setting, where d_{max} is increased from 1 to 3. Since this difference is methodologically relevant, the complete set of runs is reported in Table 5.2, whereas Table 5.3 provides condensed summaries of the main comparisons. The reported results should therefore be interpreted with this protocol difference in mind.

Motivated by the promising performance of *Diofantos* at degree 3, we also evaluated degree 4. However, runs with degree 4 exceeded the imposed time limit on almost every data set, and this setting was therefore not pursued further. For *Diofantos* with maximal degrees 3 and 4, the experiments were run exceptionally in parallel, with the time limit imposed separately on each data set, following the protocol used in the OEIS experiments. Further details are provided in Subsection 5.2.7. For SINDy, we initially considered the same degree settings and, because its computational cost was more manageable, extended the degree sweep further to maximal degrees 4, 5, and 10. Finally, *MoadeeB* was evaluated both with its default settings and with enlarged values of the magnitude, length, and `top_n` parameters, each increased by a factor of 30 to 1500, 300, and 300, respectively.

For the rational benchmark, the experimental setup adopts the best-performing configurations for each method identified on the polynomial benchmark: the default settings for *MoadeeB*, a fixed maximal degree of 3 for *Diofantos*, and a sweep over maximal degrees from 1 to 10 for SINDy. Given the substantial proportion of polynomial instances in this benchmark, we also evaluate *Diofantos* and SINDy to provide additional reference points.

Finally, for the implicit benchmark, we evaluated only *MoadeeB*, as it is the only method among those considered that is suitable for this type of equation.

5.2.4 Polynomial equations

Table 5.2 reports the success rates of all methods under the corresponding experimental settings and protocols, together with approximate wall-clock times. The settings consist of the maximal polynomial degree for *Diofantos* and SINDy, and the magnitude and length parameters for *MoadeeB*. For *Diofantos* and SINDy, the protocol indicates whether the method incrementally loops (following Sections 3.2 and 4.2) over maximal degrees or is run only at the prescribed maximal degree (indicated by *(no loop)*).

Table 5.2: Performance of *MoadeeB*, *Diofantos*, and SINDy on the synthetic polynomial benchmark under different experimental protocols and parameter settings. The table reports the number of reconstructed equations out of 2000, the corresponding success rates, and the approximate wall-clock time required for a sequential run over all data sets. Times are reported in seconds (s), minutes (min), hours (h), or days (d), as appropriate. The best configurations, which are also presented in Table 5.3, are shown in bold.

Approach	**Max. Deg.**	***mag len top_n***	**Recon.**	**Success**	**Wall-clock Time**
MoadeeB	\	**50, 10, 10**	**652**	**32.62%**	**4 min 40 s**
MoadeeB	\	1500, 300, 10	652	32.62%	36 min
MoadeeB	\	1500, 300, 300	652	32.62%	4 h
Diofantos (no loop)	1	\	114	5.7%	3 min 13 s
Diofantos (no loop)	2	\	613	30.65%	1 h 10 min
***Diofantos* (no loop)**	**3**	\	**1172**	**58.62%**	**$\geq$ 2d**
Diofantos (no loop)	4	\	1	0.05%	$\geq$ 2d
Diofantos	2	\	613	30.65%	1 h 11 min
Diofantos	3	\	799	39.95%	$\geq$ 2d
Diofantos	4	\	799	39.95%	$\geq$ 2d
SINDy (no loop)	1	\	114	5.7%	53 s
SINDy (no loop)	2	\	571	28.55%	1 min 55 s
SINDy (no loop)	3	\	38	1.90%	3 min 26 s
SINDy	2	\	577	28.85%	1 min 55 s
SINDy	3	\	599	29.95%	3 min 26 s
SINDy	4	\	639	31.95%	4 min 26 s
SINDy	5	\	672	33.6%	6 min
SINDy	**10**	\	**694**	**34.7%**	**50 min**

Overall, *Diofantos* achieves the strongest performance, reconstructing 1172 out of 2000 equations (58.62%) with its best parametrization. In comparison, SINDy reconstructs at most 694 equations (34.70%), while *MoadeeB* reconstructs 652 equations (32.62%). We also observe substantial differences in the approximate wall-clock time required for the experiments. All experiments involving *Diofantos* with maximal degree 3 or higher required more than two days when measured sequentially. The higher success rate of *Diofantos* is therefore accompanied by greater computational cost. SINDy, on the other hand, requires a smaller computational budget, making it feasible to explore more flexible

choices of maximal degree. Setting the maximal degree to 10 took almost one hour, in contrast to a maximal degree of 5, which took approximately 6 minutes. Nonetheless, this increased the success rate by only 1.1%.

Both *Diofantos* and SINDy expose a key complexity-control hyperparameter: the maximal polynomial degree permitted during the discovery process. To ensure a fair comparison while avoiding an exhaustive hyperparameter sweep, Table 5.3 reports, for each method, the single configuration that achieves the highest overall reconstruction count on this benchmark. Concretely, the best-performing configuration for *Diofantos* uses a maximal degree of 3 directly, without first trying linear and quadratic polynomials as with OEIS sequences. On the other side, SINDy attains its best score when increasing the maximal degree up to 10. In contrast, *MoadeeB*'s performance remains unaffected by the parameters controlling the filtering criteria.

Table 5.3: Summary of the performance of *MoadeeB*, *Diofantos*, and SINDy on the synthetic polynomial benchmark, grouped by the polynomial degree of the ground-truth equation. Success is measured by the number of reconstructed equations equivalent to the ground truth. Only best-performing configurations are used: default settings for *MoadeeB*, a fixed maximal degree of 3 for *Diofantos*, and, for SINDy, a sweep over maximal degrees from 1 to 10.

Degree	**Group Size**	***MoadeeB***	***Diofantos***	**SINDy**
0	13	13 (100.0%)	13 (100%)	13 (100%)
1	101	101 (100.0%)	101 (100%)	101 (100%)
2	499	452 (90.58%)	**499 (100%)**	463 (92.79%)
3	559	86 (15.38%)	**559 (100%)**	36 (6.44%)
4	391	0	0	**30 (7.67%)**
5	226	0	0	**29 (12.83%)**
6	132	0	0	**18 (13.64%)**
7	54	0	0	**4 (7.41%)**
≥ 8	24	0	0	0
Σ	2000	652 (32.62%)	**1172 (58.62%)**	694 (34.72%)

Results in the table are grouped by the polynomial degree of the ground-truth equation underlying each data set. The same degree-wise breakdown is visualized in Figure 5.1, where the dashed curve indicates the size of each degree group and the bars show the number of successful reconstructions for each method.

Notably, *Diofantos* attains a 100% success rate for all equations within the representational range induced by its degree constraint, i.e., for polynomials of degree at most 3. In contrast, both *MoadeeB* and SINDy exhibit a marked drop in performance at degree 3 or more. Nonetheless, SINDy is the only method that remains useful on the subset of equations of degree 4 or higher, where *MoadeeB* and *Diofantos* have zero reconstruction rate.

Table 5.4 presents selected examples of the polynomial equations used to generate the benchmark data sets.

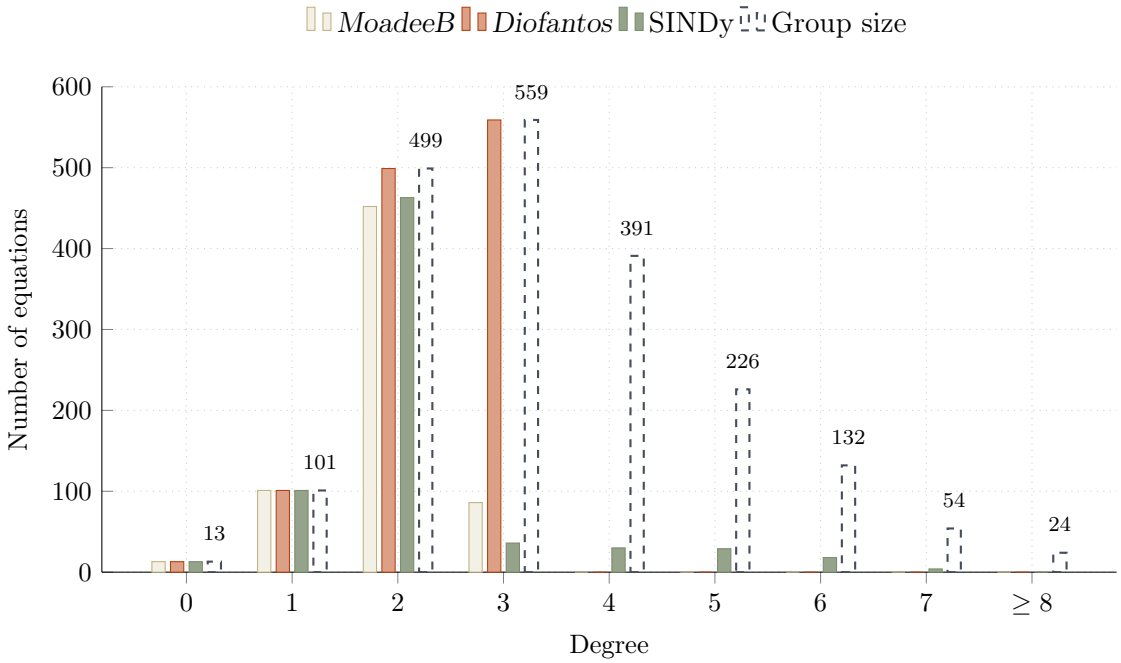


Figure 5.1: Number of correctly reconstructed polynomial equations by degree for *MoadeeB*, *Diofantos*, and SINDy. The dashed line shows the size of each degree group.

Table 5.4: Selected equations used to generate the synthetic polynomial benchmark data sets. The table reports the equation identifier, the corresponding ground-truth equation, its polynomial degree, and a reconstruction indicator. For the examples shown, all compared methods either succeeded or failed together.

ID	Equation	Degree	Reconstructed
0	target $= 3$	0	Yes
1	target $= -9vy - 2x + 6$	2	Yes
2	target $= 5v + 2$	1	Yes
3	target $= -v + 5xy^2z$	4	No
4	target $= w$	1	Yes
5	target $= 8xy$	2	Yes
6	target $= -4vz$	2	Yes
7	target $= 6z$	1	Yes
8	target $= -6v - w$	1	Yes
⋮			
1999	target $= -7x^2yz^2$	5	No

5.2.5 Rational equations

Table 5.5 reports the results of the computational evaluation of the *MoadeeB*, *Diofantos*, and SINDy on the benchmark consisting of 1000 equations generated by the rational grammar. In terms of aggregate success rate, *MoadeeB* performs best overall, reconstructing 479 out of 1000 equations (47.9%), followed by *Diofantos* with 384 successful reconstructions (38.4%) and SINDy with 254 (25.4%). These aggregate results, however, conceal

substantially different performance profiles across degrees.

Table 5.5: Performance of *MoadeeB*, *Diofantos*, and SINDy on the rational benchmark, grouped by the degree of the corresponding implicit polynomial equation. Success is measured by the number of reconstructed equations equivalent to the ground truth. The final row reports the number of polynomial equations identified from integer-valued data sets. Best-performing configurations identified on the polynomial benchmark are adopted: default settings for *MoadeeB*, a fixed maximal degree of 3 for *Diofantos*, and, for SINDy, a sweep over maximal degrees from 1 to 10.

Degree	**Group Size**	***MoadeeB***	***Diofantos***	**SINDy**
0	6	6 (100%)	6 (100%)	6 (100%)
1	73	**73 (100%)**	57 (78.08%)	57 (78.08%)
2	334	**323 (96.71%)**	173 (51.8%)	156 (46.71%)
3	285	76 (26.67%)	**148 (51.93%)**	15 (5.26%)
4	177	1 (0.56%)	0	**13 (7.34%)**
5	67	0	0	**1 (1.49%)**
6	32	0	0	**2 (6.25%)**
7	20	0	0	**4 (20%)**
≥ 8	6	0	0	0
Σ	1000	**479**	384	254
Polynomials	545	251 (46.06%)	**384 (70.46%)**	254 (46.61%)

To enable a consistent notion of complexity across rational expressions, we group results by the polynomial degree of the *implicit polynomial form* associated with each rational equation. Concretely, given a rational equation with the target variable isolated on the left-hand side, we multiply both sides by the denominator and rearrange terms to obtain an equivalent implicit equation of the form $f(\mathbf{x}) = 0$. The reported degree is the total degree of this resulting polynomial. For example, the rational equation $y = x/z^2$ is assigned degree 3, since its associated implicit form is $yz^2 - x = 0$.

The same degree-wise breakdown is visualized in Figure 5.2, where the dashed curve indicates the size of each degree group and the bars show the number of successful reconstructions for each method.

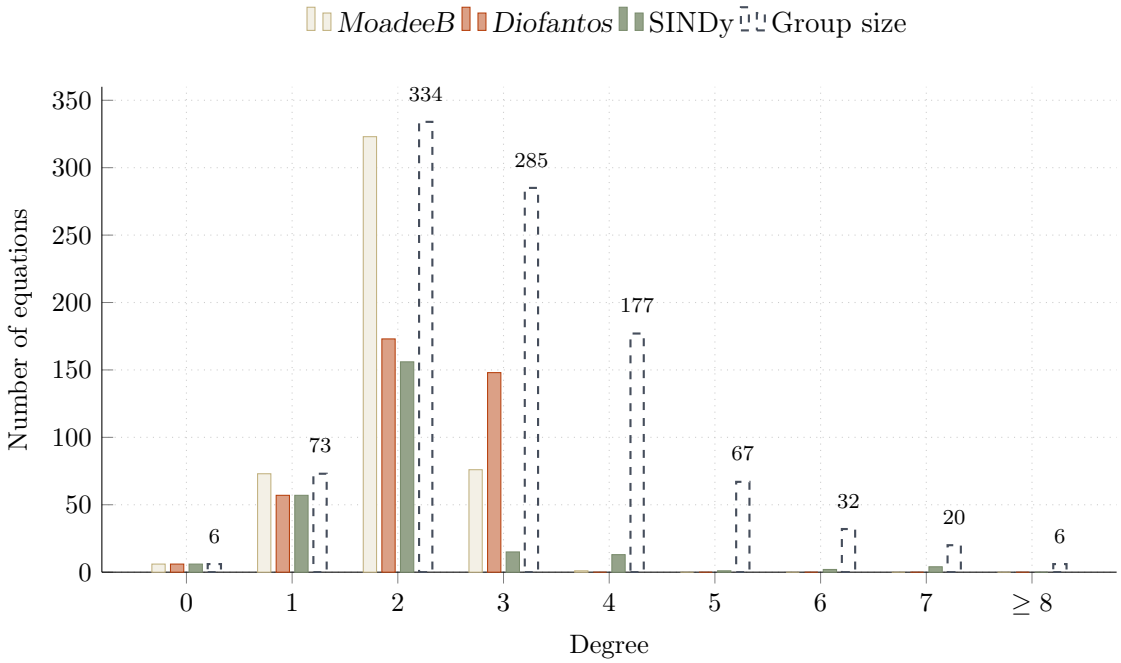


Figure 5.2: Number of correctly reconstructed rational equations by degree for *MoadeeB*, *Diofantos*, and SINDy. The dashed line shows the size of each degree group.

MoadeeB performs best on low-degree instances. It achieves perfect reconstruction for degrees 0 and 1, and remains very strong for degree 2, where it reconstructs 323 out of 334 equations (96.71%). Its performance then drops sharply at degree 3 to 26.67% (76 out of 285), and only one successful reconstruction is obtained for degrees ≥ 4. The group of equations with degree at least 4 contains 302 instances in total, namely 177 of degree 4, 67 of degree 5, 32 of degree 6, 20 of degree 7, and 6 of degree at least 8.

Diofantos shows a different behavior. While its performance is below that of *MoadeeB* for degrees 1 and 2, it achieves the strongest result at degree 3, reconstructing 148 out of 285 equations (51.93%), and thus clearly outperforming both *MoadeeB* and SINDy in this group. As expected from its restriction to polynomial models of maximal degree 3, it does not recover any equations of degree 4 or higher. SINDy performs similarly to *Diofantos* at degrees 0 and 1, somewhat worse at degree 2, and substantially worse at degree 3, where it reconstructs only 15 equations (5.26%). On the other hand, unlike *MoadeeB* and *Diofantos*, it still achieves a small number of successful reconstructions for higher degrees, namely 13 at degree 4, 1 at degree 5, 2 at degree 6, and 4 at degree 7.

The final row of Table 5.5 further shows that all successful reconstructions of *Diofantos* and SINDy occur within the subset of 545 integer-valued data sets generated from polynomial equations. In contrast, *MoadeeB* reconstructs 251 equations in this subset, which indicates that a substantial part of its overall success comes from instances beyond those accessible to the two reference methods.

Table 5.6 presents selected examples of the equations used to generate the rational benchmark data sets, together with information on which methods successfully reconstructed them.

5.2.6 Implicit equations

Table 5.7 summarizes the results of the computational evaluation of *MoadeeB* on the benchmark of 20 implicit polynomial equations. The results are grouped by polynomial

Table 5.6: Selected equations used to generate the rational benchmark data sets. The table reports the equation identifier, the corresponding ground-truth equation, its implicit polynomial degree, and the methods that successfully reconstructed an equivalent equation.

ID	Equation	Degree	Reconstructed by
0	target $= 3vw^2xyz^2$	7	None
1	target $= -9v - 2$	1	All
2	target $= (2v - 1)/(6v + 5)$	2	*MoadeeB*
3	target $= 5v + xy^2z$	4	None
4	target $= 8w - 4$	1	All
5	target $= 6$	0	All
6	target $= (-wy - 6)/(-6)$	2	*MoadeeB*
7	target $= 9 - 7y$	1	All
8	target $= -2w$	1	All
⋮			
999	target $= 7vwxz - 5xyz^2$	4	None

degree.

Table 5.7: Performance of *MoadeeB* on the implicit benchmark, grouped by polynomial degree. Success is measured by the number of reconstructed equations equivalent to the ground truth.

Degree	Group Size	Success	Success Rate
2	3	3	100%
3	3	1	33.3%
4	3	2	66.6%
5	7	0	0%
7	3	0	0%
8	1	0	0%
Σ	20	6	30%

MoadeeB successfully reconstructed all implicit equations of degree 2. For higher degrees, the success rate decreases: one equation of degree 3 and two equations of degree 4 were correctly reconstructed. No successful reconstructions were obtained for polynomials of degree 5, 7, or 8.

An interesting observation is that the method reconstructed more equations of degree 4 than of degree 3, although a higher polynomial degree would generally be expected to make the reconstruction task more difficult. However, given the small number of instances in these groups, we do not interpret this difference as indicating a meaningful trend. Manual inspection of the corresponding equations did not reveal any obvious structural property that could explain this behavior.

Overall, the method reconstructed 6 out of the 20 implicit equations, corresponding to an overall success rate of 30%.

Table 5.8 presents the implicit polynomial equations included in the benchmark, together with their reconstruction status.

Table 5.8: Implicit equations used to generate the implicit benchmark data sets. The table reports the equation identifier, the corresponding ground-truth equation, its degree, and whether *MoadeeB* successfully reconstructed an equivalent equation.

ID	Equation	Degree	Reconstructed
0	$-7x^2y^2 = 0$	4	Yes
1	$-6x^2 + 10y^2 - 4 = 0$	2	Yes
2	$-6x^2y^3 - 6 = 0$	5	No
3	$7x^2 - 4z^2 + 4 = 0$	2	Yes
4	$-8x^3y^2z^2 + 9x - 9 = 0$	7	No
5	$-10y^2 + 2z^2 + 8 = 0$	2	Yes
6	$-3y^2z^3 + 3 = 0$	5	No
7	$7x^3z^2 - 7 = 0$	5	No
8	$-8x^2y^3z^2 - 2 = 0$	7	No
9	$x^2 + 5xz^2 + 6 = 0$	3	No
10	$-y^2z - 9z^2 + 10 = 0$	3	Yes
11	$10x^3y^2 + 10 = 0$	5	No
12	$-3x^2 + 4y^2z^2 + 9yz + 2 = 0$	4	Yes
13	$6x^2z^3 - 6 = 0$	5	No
14	$8y^3z^2 - 2 = 0$	5	No
15	$-5x^4z^3 + 10 = 0$	7	No
16	$-10x^3y^2 - x - 7 = 0$	5	No
17	$2x^2z - 10z^2 + 10 = 0$	3	No
18	$-7x^2y^4z^2 + 7z + 7 = 0$	8	No
19	$-9y^3 + 4y^2z^2 + 5 = 0$	4	No

5.2.7 Running times

Approximate wall-clock time equivalents of sequentially run experiments on the polynomial benchmark are reported in Table 5.2. With the exception of *Diofantos*, all experiments completed smoothly and terminated within two hours. *Diofantos* was originally configured with a maximal degree of 3. As discussed in Section 3.1.5, allowing degree 3 induces a combinatorial growth in the number of candidate terms (columns) considered by the method. This substantially increases the size and difficulty of the resulting Diophantine system, and consequently the time required by the underlying solver. Aside from the size of the Diophantine system, higher degrees naturally mean also larger values in the target column, which is also one of the factors that determine the time complexity of the Diophantine solver.

In the initial set of experiments, *Diofantos* with the maximal degree of 3 was executed sequentially and required more than 24 hours to complete. Since the maximal-degree 2 configuration needed less than two hours and achieved a 100% success rate on all polynomial instances of degree at most 2, we nevertheless repeated the degree 3 experiment under a more appropriate computational budget. Specifically, we enabled parallel execution for these experiments, with each data table processed independently on a separate computational resource, and imposed a two-day time limit per equation, following the protocol used in the OEIS experiments. Under this configuration, the hardest (unsuccessful) instances reached the full time limit, whereas each run that successfully recovered the ground-truth equation terminated in under three minutes. Consequently, for these runs, Table 5.2 reports more than 2 days of runtime. This information motivated us to speed

up the experiments and reduce the time limit per data set from two days to half an hour. The surprisingly high 100% recovery success rate for polynomials of degree 3 motivated us to extend the experiments to polynomials of degree 4. This time, the only equation discovered in under half an hour was the zero equation $y_{target} = 0$, which *Diofantos* found in 15 minutes.

Overall, these results indicate that *Diofantos* is the most effective method on the polynomial benchmark when the target equation is known to be polynomial and the maximal degree can be bounded a priori. However, when degree 3 or higher is permitted, we recommend using parallelization together with a time limit in order to balance reconstruction performance and computational cost. However, wall-clock time is not an appropriate measure of theoretical time complexity, so we report running times only to provide a general indication of the practical computational cost of our implementations.

The rational benchmark has a similar runtime profile.

5.2.8 Discussion

Across both the polynomial and rational benchmarks, *MoadeeB* exhibits a consistent degree-dependent performance pattern: it achieves perfect reconstruction for equations of degree at most 1, remains strong for degree 2, and drops sharply for degree 3 (approximately 15% success). This behavior is expected for equation discovery methods in general, given the rapid growth of the hypothesis space with degree and the resulting increase in ambiguity when only a limited number of samples is available.

On the polynomial benchmark, we hypothesize that *MoadeeB* fails on higher-degree ground-truth equations primarily due to overfitting by lower-degree polynomials. With relatively few data points (rows), it is often possible to interpolate the observations using lower-degree models that nevertheless involve many nonzero terms and large coefficients. Such interpolants can match the sampled data exactly while deviating from the ground truth as symbolic expressions, thereby preventing recovery of the target equation.

This hypothesis can be motivated by the operation of Möller–Buchberger algorithm described in Section 4.1.2. The algorithm considers monomials according to the chosen monomial ordering and terminates once it has identified a basis of m linearly independent evaluation vectors, where m is the number of data points. Consequently, a sufficiently large set of lower-degree monomials may allow the algorithm to construct the complete basis before reaching higher-degree monomials that occur in the ground-truth equation. In our setting, the equations involve six variables, including the target variable. There are 28 monomials of degree at most two, 84 of degree at most three, and 210 of degree at most four. Thus, with only 20 data points, it is possible for Möller–Buchberger algorithm to obtain 20 linearly independent evaluation vectors using monomials of degree at most two, before reaching monomials of degree three. Similarly, with 84 data points, the complete basis may in principle be obtained using monomials of degree at most three, before any degree-four monomials are reached. This provides a mechanism by which the rapid growth in the number of monomials with increasing degree can hinder the recovery of higher-degree ground-truth equations.

To test this explanation directly, we conducted a small supplementary experiment using one degree-3 equation, $u = -6xzw$, and one degree-4 equation, $u = -6xzw^2$. For each equation, we generated data sets containing 20, 60, and 100 observations, repeating each setting with three different random seeds, for a total of nine data sets per equation. In all repetitions with 60 or 100 observations, Möller–Buchberger algorithm successfully reconstructed the ground-truth equation, whereas it failed in all repetitions with 20 observations. These results provide direct, although limited, empirical support for the hypothesis that increasing the number of observations can prevent premature reconstruction based on lower-

degree interpolating relations and thereby facilitate recovery of higher-degree ground-truth equations.

Our explanation for the unchanged behavior of *MoadeeB* despite relaxing the filtering criteria is based on qualitative inspection of the candidate equations generated across the experiments. Although this inspection was not exhaustive, a recurring pattern was a pronounced gap between equations of low and reasonable complexity and highly complex, apparently overfitted equations. Moreover, the number of low-complexity candidates was typically very small, often limited to only a handful and, in many cases, to a single equation.

The results obtained with *Diofantos* are promising for lower maximal degrees, but they also indicate that increasing the maximal degree is not straightforwardly beneficial. In particular, experiments with maximal degree 4 produced worse results: the computational cost became prohibitively large, and almost no positive case finished within the thirty-minute runtime limit per data set.

More generally, reconstruction failures may reflect insufficient observations, the maximal degree chosen for *Diofantos*, or computational constraints, and the present experiments do not fully disentangle these factors. Although *MoadeeB* supports a broader class of relations, its reconstruction performance also declines at higher polynomial degrees under the evaluated settings, with limited observations potentially hindering recovery. Discovering more complex mathematical relations therefore remains challenging for both methods.

Overall, the results suggest that *Diofantos* is often the most effective approach when the target relation is expected to be a low- or medium-degree polynomial equation. At the same time, its computational cost can become prohibitive as the maximal degree increases, making parallelization and explicit time limits advisable in practice. In contrast, *MoadeeB* remains the more broadly applicable method: it naturally supports rational and implicit polynomial equations, which are outside the scope of *Diofantos*, and therefore provides a more general-purpose baseline despite its reduced performance on higher-degree polynomials. SINDy, however, can serve as a complementary tool for discovering even higher-degree polynomial relations in settings where both *MoadeeB* and *Diofantos* become ineffective or computationally infeasible.

The results of the evaluation presented in this chapter support hypothesis H3: while no single method dominates across all equation classes, the developed methods compare favorably with existing approaches across a substantial subset of exact equation discovery tasks.

Chapter 6

Conclusions

This dissertation set out to address the gap between equation discovery and experimental mathematics. To that end, it introduced the task of exact equation discovery, a form of equation discovery concerned not with approximate relations, but with equations that fit the observed data perfectly. This task is motivated by the desire to discover mathematical knowledge from mathematical data, stored in databases such as the On-Line Encyclopedia of Integer Sequences (OEIS), where the goal is not merely to model the data well, but to recover exact mathematical structure. Methods tailored to this setting are, to the best of our knowledge, either absent or significantly underexplored. Existing approaches typically either operate in approximate settings, as in symbolic regression, or rely on stronger domain-specific assumptions, as in neural symbolic inference [34], [38] and *guessing* methods from experimental mathematics [18].

To address this gap, we developed two methods grounded in number theory and computational algebraic geometry. The first, *Diofantos*, is a novel algorithm for discovering exact equations directly from observed data. It builds on the well-established SINDy (Sparse Identification of Nonlinear Dynamics) framework for equation discovery [25] and on a recent variant of Lagrange [59], but departs from these predecessors by replacing (sparse) linear regression with a solver for systems of linear Diophantine equations in order to identify optimal non-linear combinations of terms. The second method, *MoadeeB*, targets implicit polynomial and rational equations through the computation of Gröbner bases. It relies on the connection between equation discovery and the algebro-geometric concepts of varieties, vanishing ideals, and Gröbner bases. We evaluated both methods on more than 30,000 integer sequences from the OEIS and on more than 3,000 synthetic equations, and compared them with SINDy as well as with methods specialized for integer sequences.

The results show that exact equation discovery is a meaningful and feasible task, and that the two proposed methods are well suited to it. The methods are also complementary. *Diofantos* performs best on polynomial equations of degree ≤ 3, whereas *MoadeeB* is better suited to a broader class of implicit and rational equations. In exact settings, both methods compare favorably with SINDy, and in several scenarios they also outperform methods specialized for integer sequences. Their practical value is further supported by the discovery of one simplified recurrence and two previously unknown recurrence relations for OEIS sequences, which were subsequently approved by OEIS curators; two of these equations concern sequences regarded by the OEIS as important and well known.

6.1 Validity of the Hypotheses Proposed by the Dissertation

After summarizing and reviewing the contents of this dissertation, we can now examine the validity of the hypotheses set at the beginning of this work.

6.1.1 Hypothesis 1

We can design novel frameworks for the discovery of exact equations based on solvers for Diophantine equations and the computation of Gröbner bases.

Confirmed. We designed two novel frameworks for the discovery of exact equations: one based on solvers for Diophantine equations and one based on the computation of Gröbner bases.

In number theory, solvers for systems of linear Diophantine equations are algorithms designed to find integer solutions to linear systems whose coefficients are integers. Such solvers can therefore be applied to integer tabular data in order to discover exact linear equations relating a target column to the remaining columns. By constructing additional columns corresponding to higher-order monomials, this approach can be extended from the discovery of linear equations to the discovery of polynomial equations.

As a central tool in computational algebraic geometry, a Gröbner basis can be understood as a simplified and structured set of polynomial equations that hold for the given data. For this reason, it provides a natural set of candidates for the underlying polynomial relations present in the data. Thus, computing Gröbner bases from exact data provides a viable approach to exact equation discovery.

Together, these two frameworks demonstrate that exact equation discovery can be formulated as a problem in symbolic computation. The framework based on Diophantine solvers addresses the discovery of exact polynomial relations over integer data, while the Gröbner-basis-based framework provides a complementary approach for discovering rational and implicit polynomial relations over rational-valued data.

6.1.2 Hypothesis 2

We can implement algorithms based on the designed frameworks in a software tool that enables the automated discovery of exact equations from mathematical data.

Confirmed. We successfully implemented all exact equation discovery approaches and algorithms introduced in this dissertation in two open-source Python libraries, *Diofantos* and *MoadeeB*. The empirical experiments in Chapter 3 demonstrate that constructing additional columns corresponding to higher-order monomials of the input variables, using solvers for systems of linear Diophantine equations to express the target column as an integer linear combination of other columns, and interpreting the resulting relation as a polynomial equation, provides an approach capable of discovering exact equations from mathematical data. The empirical experiments in Chapter 4 demonstrate that exact equation discovery can also be achieved by computing Gröbner bases from rational-valued tabular inputs and extracting suitable low-complexity polynomial relations from the resulting bases. Finally, the experiments involving implicit and rational equations in Chapter 5 demonstrate that *MoadeeB* is also capable of discovering implicit polynomial equations and rational equations from exact data.

6.1.3 Hypothesis 3

The developed algorithms compare favorably with existing approaches in terms of performance and applicability across a substantial subset of exact equation discovery tasks.

Confirmed.

The evaluation shows that exact-arithmetic equation discovery compares favorably with approximate approaches when the target relation lies within the supported hypothesis classes. On the OEIS linear-recurrence and core benchmarks, Diofantos and MoadeeB generally reconstructed more exact relations than SINDy, while the synthetic experiments

confirmed a clear advantage of Diofantos for low-degree polynomial equations and of MoadeeB for low-degree rational relations; SINDy became comparatively stronger for polynomial equations of degree four or higher. Comparisons with specialized OEIS methods were more mixed: QSynt benefited from the expressiveness of program synthesis on heterogeneous core sequences, Guessing was particularly effective for its specialized class of linear recurrences, and TM-OEIS was stronger in some one-step prediction settings, whereas the exact methods generally performed better for longer prediction horizons. Unlike these sequence-specific approaches, however, Diofantos and MoadeeB were also successfully applied to general exact tabular data outside the OEIS, supporting the broader applicability claimed in Hypothesis 3.

The experiments also demonstrate that Diofantos and MoadeeB have complementary strengths rather than one uniformly dominating the other. Diofantos is particularly effective for explicit polynomial equations over integers when the maximal degree can be bounded in advance, and it reconstructs substantially more documented linear recurrences than MoadeeB; its main limitations are the rapid increase in computational cost with polynomial degree and its restriction to explicit integer relations. MoadeeB extends the applicable equation class to rational-valued data, rational coefficients, and implicit polynomial relations, which explains its stronger performance on the core OEIS and rational benchmarks and its applicability to the implicit benchmark; it also exhibited substantially shorter running times in the OEIS experiments. Conversely, its unrestricted search often returns nonlinear relations rather than the documented linear recurrence, and its reconstruction performance decreases markedly as the polynomial degree increases. These results establish the two approaches as complementary tools: Diofantos is especially effective for bounded-degree explicit polynomial discovery, whereas MoadeeB provides greater representational generality.

6.1.4 Hypothesis 4

The developed approaches are able to discover new, previously unknown mathematical knowledge in the form of exact equations.

Confirmed. The application of the developed methods led to the identification of several previously unknown recurrence relations within the OEIS, demonstrating the practical discovery potential of the proposed exact equation discovery frameworks. In total, we discovered one accepted simplification of a known recurrence and two genuinely novel nonlinear recurrences, all of which were subsequently verified, as detailed in Subsections 3.2.2 and 4.2.3. These findings not only provide empirical support for the theoretical approach but also constitute genuine contributions to the mathematical knowledge collected in the OEIS database.

Using *Diofantos* and SINDy, we obtained a simplified form of a previously known recurrence relation for the integer sequence with OEIS identifier A078475. The newly derived recurrence,

$$a_n = -9a_{n-2} - 36a_{n-4} - 84a_{n-6} - 126a_{n-8} - 126a_{n-10} - 84a_{n-12} - 36a_{n-14} - 9a_{n-16} - a_{n-18},$$

was submitted to and formally accepted into the corresponding OEIS entry A078475 [106]. With *MoadeeB*, we extended these results by uncovering two entirely new nonlinear recurrence relations. For the sequence of binary tree counts by height (OEIS A001699), we identified the following recurrence:

$$a_n = -a_{n-2}^3 + a_{n-1}^2 + 3a_{n-1}a_{n-2} + 2a_{n-2}^2 + 2a_{n-1} - 4a_{n-2}.$$

Similarly, for the sequence of planted 3-tree counts (OEIS A002658), we discovered the relation:

$$a_n = -\frac{a_{n-2}^3 - 2a_{n-1}^2 - 6a_{n-1}a_{n-2} - 4a_{n-2}^2 - 8a_{n-1} + 11a_{n-2}}{4}.$$

These two previously unknown equations have since been rigorously proved and formally accepted into the corresponding OEIS entries, A001699 and A002658 [114], [115].

6.2 Scientific Contributions

This PhD dissertation has advanced the field of equation discovery by developing new methods for identifying exact mathematical equations directly from data. The proposed approaches integrate established principles from symbolic regression, number theory, and computational algebra. A more detailed discussion of its contributions to science is given below.

C1 **The first major contribution lies in the innovative application of solvers of Diophantine equations to the problem of exact equation discovery. These solvers enable the automated identification of exact equations that inherently hold for data generated within mathematical domains. Because the discovered equations must satisfy the underlying integer constraints by design, they possess intrinsic validity and reliability, thereby enhancing the robustness of the equation discovery process.** The importance of this contribution lies in extending equation discovery beyond its usual approximate setting toward exact symbolic recovery. To the best of our knowledge, approaches specifically designed for this setting remain scarce, even though such methods are highly relevant for mathematical data. While approximate methods may sometimes be adapted to exact-valued inputs, they are not formulated to exploit exact arithmetic and therefore do not provide the same degree of reliability in recovering exact equations. In contrast, the approach introduced in this dissertation reformulates the problem as the solution of systems of linear Diophantine equations, thereby providing a principled method for discovering exact polynomial relations from integer-valued tabular data.

C2 **The second contribution involves employing Gröbner basis computation to enable the discovery of exact polynomial equations. This approach facilitates the derivation of polynomial and rational relationships involving rational coefficients, eliminating the need for manual construction of higher-degree features. Consequently, this method expands the range of problems addressable by equation discovery systems while improving both their computational efficiency and feasibility. By leveraging algebraic methods, the approach ensures mathematically sound results.** This contribution complements the method based on Diophantine solvers by extending exact equation discovery from integer-valued data to rational-valued data. It also broadens the class of discoverable relations: rather than being restricted primarily to polynomial equations with integer coefficients, the proposed approach enables the discovery of polynomial and rational relations involving rational coefficients. In addition, Gröbner basis computation makes it possible to discover implicit polynomial equations without manually constructing higher-degree features in advance. This algebraic formulation also addresses one of the main practical limitations of *Diofantos*, namely its computational cost, and

thereby improves the feasibility of exact equation discovery for a wider range of problems.

C3 **The final contribution reframes the discovery of recurrence relations in integer sequences as an exact equation discovery task. Within this framework, the algorithms developed in this dissertation were applied to identify new, previously unknown recurrence relations in integer sequences catalogued in the On-Line Encyclopedia of Integer Sequences (OEIS). Three novel equations were discovered and subsequently proved to hold, demonstrating the capability of the proposed methods to contribute new mathematical knowledge.** This contribution demonstrates that exact equation discovery can be used not only to recover known mathematical structures, but also to generate new mathematical results. By interpreting consecutive terms of an integer sequence as exact data and treating recurrence relations as equations among shifted sequence values, the problem of recurrence discovery is placed within the same algebraic framework as the other equation discovery tasks studied in this dissertation. This reformulation makes it possible to apply the developed methods to sequences for which no recurrence relation is known in advance. The discovery and proof of three new recurrence relations for OEIS sequences provide concrete evidence that the proposed methods can support computer-assisted mathematical exploration. In this sense, the contribution goes beyond methodological development: it shows that exact equation discovery can serve as a practical instrument for extending existing mathematical knowledge.

6.3 Limitations

The research presented in this dissertation has several important limitations. First, it focuses on tabular data. Mathematical data and mathematical knowledge, however, are often richer than this and may involve objects such as graphs or more structured relational forms. Namely, the primary goal of much of mathematical research is the discovery of proofs rather than equations, and the present work does not directly address that broader aim. Second, the scope of the proposed methods is limited to polynomial, rational, and implicit equations, which reflects their current design and capabilities. Third, in the specific setting of the OEIS and the *linrec* data set, our methods must be viewed against a strong body of existing work in computational algebra devoted to discovering or *guessing* recurrences for integer sequences. For linear recurrences, there are also theoretical guarantees of reconstruction given sufficiently many sequence terms. Our approach does not currently provide comparable guarantees, which is an important limitation, especially in a setting where specialized methods may in principle achieve perfect reconstruction. The empirical evaluation of the methods, presented in the thesis, does not reveal whether the reconstruction failures arise from insufficient observations, the selection of candidate equations, or computational constraints. Finally, although the empirical evaluation is extensive, it could be broadened further by including additional state-of-the-art baselines, larger benchmarks, and more varied forms of analysis.

6.4 Further Work

Limitations discussed in this section also point to several promising directions for future research. One direction is to expand exact equation discovery beyond the present setting by

investigating richer mathematical data and additional model classes. Foundation models could also be explored as tools for this task, either as baselines or as components within broader hybrid systems. For instance, systems such as the agentic AI framework proposed in [121] or AI-Descartes [122] could be combined with our tools to help verify, refute, and contextualize the candidate equations produced by exact equation discovery in light of existing results in the literature, thereby reducing the reliance on manual verification.

Another relevant direction is the continued development of transformer-based symbolic regression for integer sequences, represented in this thesis by TM-OEIS [34]. More recently, Zhu et al. [123] proposed a progressive self-learning approach that adapts this paradigm more directly to the distribution of OEIS sequences. Their method iteratively generates candidate formulas, verifies them against the corresponding sequences, and incorporates successful candidates into subsequent training, with the aim of progressively improving symbolic equation discovery on the target domain. The authors also report newly discovered candidate formulas for several OEIS sequences, including sequences classified as *hard*. These results further illustrate the potential of learned symbolic-regression methods as complementary tools for equation discovery. At the same time, candidate equations inferred from a small number of known sequence terms should be treated cautiously, since exact agreement on the available terms does not by itself establish that the inferred equation represents the underlying sequence. Some caution is also warranted when interpreting the quantitative results reported by Zhu et al. In the comparison most relevant to our setting, the reported performance values of their proposed method coincide exactly with the corresponding values reported in TM-OEIS [34] for a different model. This makes these particular results difficult to interpret without independent reproduction.

Another direction is to apply exact equation discovery to other public mathematical databases, such as databases of graphs and knots [124], both as new application domains and as sources of benchmarks for evaluating *Diofantos*, *MoadeeB*, and related methods. A particularly interesting extension would be to move beyond equations derived from a single OEIS sequence and instead search for relations among different OEIS sequences. In such a setting, machine learning methods could first be used to identify strong candidate relationships among many sequences, in the spirit of Davies et al. [11], after which equation discovery methods could be applied to formulate and test conjectural exact relations. This could provide a more automated route from large-scale sequence data to new mathematical hypotheses and discoveries.

There are also several natural method-specific extensions. For *Diofantos*, one of the most promising directions would be to exploit the kernel computed by the HNF-based solver [95] to discover implicit polynomial equations and, where possible, explicit rational equations, following the example of *MoadeeB*. As stated at the end of Subsection 3.1.3, *Diofantos* currently returns only the first solution from the list produced by the Python package *Diophantine* [97]. However, a more thorough recent inspection of the package implementation revealed that, at an intermediate stage, the kernel of the corresponding linear system is computed and used to enumerate solutions whose chosen vector norm does not exceed that of a particular solution. This observation suggests two promising extensions. First, all solutions already returned by the package, or additional solutions obtained through further enumeration, could be considered, with each giving rise to a candidate equation. Second, and more substantially, the kernel could be used to enumerate nontrivial solutions of the homogeneous system, thereby enabling the discovery of implicit polynomial equations and, where possible, explicit rational equations. Such an extension would bring *Diofantos* closer in scope to *MoadeeB*.

The efficiency of *Diofantos* could also be improved by developing heuristics for selecting the number of table rows or sequence elements used as input, based, for example, on

their magnitude and on the trade-off between computational feasibility and the probability of success. Another possible direction would be to investigate alternative, potentially faster implementations of the Havas algorithm for solving systems of linear Diophantine equations.

Changing the solver's default LLL-reduction parameter, $\alpha = \frac{3}{4}$, is another possibility that was not investigated in this work. This parameter controls the trade-off between reducing the magnitude of intermediate matrix entries and the number of computational steps. More generally, alternative solvers for systems of linear Diophantine equations, including other HNF-based approaches, could also be investigated. Different solvers may exhibit different trade-offs between the size of outputted solutions and the number of computational steps required for termination. Finally, as an alternative to solving systems of linear Diophantine equations, future work could consider solvers for systems of linear equations over the rationals [96]. Such an extension would address a current limitation of *Diofantos*, namely that it accepts only integer-valued input tables.

It would be interesting to evaluate the Berlekamp–Massey algorithm on the *linrec* benchmark, although theoretical results already provide substantial insight into its expected performance. In particular, for a sequence with minimal recurrence order L, at most $2L$ initial terms are sufficient to guarantee reconstruction of the minimal recurrence under the usual assumptions. Comparing the number of available sequence terms with the documented recurrence orders would therefore provide a lower-bound estimate of the expected success rate of Berlekamp–Massey on *linrec*. This estimate would nevertheless remain approximate. Some sequences may admit recurrences of lower order than those documented in the source, and Berlekamp–Massey may recover the correct recurrence even when the sufficient conditions of the theoretical guarantee are not met. In contrast, the achievable success rate may drop below this lower bound for sequences whose running time exceeds the available computational budget. Running Berlekamp–Massey directly, potentially in combination with other Guessing algorithms, would therefore provide a more realistic estimate of the achievable performance ceiling on the *linrec* benchmark.

For *MoadeeB*, it would be worthwhile to evaluate close alternatives to the Möller–Buchberger algorithm [71], [72] in the context of exact equation discovery, since these variants may improve performance in settings with many variables and relatively few data points [72].

Experiments with larger numbers of examples, i.e., more rows in the input data tables, could also be considered, motivated by the hypothesis discussed in Subsection 5.2.8 that higher polynomial degrees require more observations for reliable reconstruction. Accordingly, the synthetic benchmark could be repeated using data sets with larger numbers of sampled points. For integer sequences, a similar extension would involve using more sequence elements instead of the current heuristic of selecting $2 \cdot o + 10$ elements as a function of the recurrence order o.

More generally, systematic experiments with both *Diofantos* and *MoadeeB* could assess how the number of input observations affects reconstruction rates and computational cost. These findings could guide the development of heuristics for selecting suitable input sizes.

Particularly interesting would be an in-depth theoretical study of the underlying algorithms based on Havas solver and Möller–Buchberger algorithm and the field-specific heavy theory that supports them.

Overall, this dissertation shows that exact equation discovery can serve as a meaningful bridge between equation discovery and experimental mathematics, with promise not only for reconstructing known mathematical relations but also for generating new hypotheses and enriching mathematical databases such as the OEIS.

Appendix A

Pseudocode of *Diofantos* Experimental Setups

Algorithms A.1, A.2, and A.3 in this section provide the full details of the experimental setup used to evaluate *Diofantos* and SINDy on the *linrec* and *core* data sets, as described in Section 3.2.

Algorithm A.1: Evaluating *Diofantos* on the *core* data set

Input: List A of the first 200 elements of a sequence a_i.

Output: Recursive polynomial equation $a_n = q(n, a_{n-1}, \ldots, a_{n-10})$

```
for d_max ∈ {1, 2, 3} do
    for o ∈ {0, 1, 2, ..., 10} do
        Construct observation matrix [y|X] from the elements of a_i, corresponding
         to variables V = {a_n, a_{n-1}, ..., a_{n-o}, n}.
        if Diofantos([X|y], d_max) returns an equation e and order(e) = o then
            return e
        end
    end
end
return Failed
```

Algorithm A.2: Evaluating *SINDy-tuned* on the *linrec* data set

Input: List A of the first 200 elements of a sequence a_i.

Output: Linear-recursive equation $a_n = c_0 + \sum_{i=1}^{20} c_i a_{n-i}$

$A = \text{Truncate}(A, d_{\max} = 1)$
for $o \in \{1, 2, \ldots, 20\}$ **do**
  Construct observation matrix $[\boldsymbol{y}|X]$ from the list A, corresponding to variables $V = \{a_n, a_{n-1}, \ldots, a_{n-o}\}$.
  for $\theta \in \{0, 0.1, 0.2, 0.3, \ldots, 0.9\}$ **do**
    for *ens* $\in$ {*True, False*} **do**
      $e = \text{SINDy}([X|\boldsymbol{y}], d_{max} = 1, \theta = \theta, \text{ens} = \text{ens})$
      if *Validate(e)* **then**
        return $e : a_n = c_0 + \sum_{i=1}^{o} c_i a_{n-i}$
      end
    end
  end
end
return Failed

Algorithm A.3: Evaluating *SINDy-tuned* on the *core* data set

Input: List A of the first 200 elements of a sequence a_i.

Output: Recursive polynomial equation $a_n = q(n, a_{n-1}, \ldots, a_{n-10})$

for $d_{max} \in \{1, 2, 3\}$ **do**
  $A = \text{Truncate}(A, d_{\max})$
  for $o \in \{0, 1, 2, \ldots, 10\}$ **do**
    Construct observation matrix $[\boldsymbol{y}|X]$ from the list A, corresponding to variables $V = \{a_n, a_{n-1}, \ldots, a_{n-o}, n\}$.
    for $\theta \in \{0, 0.1, 0.2, 0.3, \ldots, 0.9\}$ **do**
      for *ens* $\in$ {*True, False*} **do**
        $e = \text{SINDy}([X|\boldsymbol{y}], d_{max} = 1, \theta = \theta, ens = ens)$
        if *Validate(e)* **then**
          return e
        end
      end
    end
  end
end
return Failed

Appendix B

Pseudocode of *MoadeeB* Experimental Setups

Full details for the experimental setup for evaluating *MoadeeB* on the *linrec* and *core* sequences are available in Algorithm B.1 and Algorithm B.2, respectively. We utilize Möller–Buchberger algorithm as implemented in the function `IdealOfPoints` of the CoCoA software [70], [113].

Algorithm B.1: Evaluating *MoadeeB* on the *linrec* data set

Input: List A of the first 200 elements of a sequence a_i.

Output: A linear-recursive equation of the form $a_n = c_0 + \sum_{i=1}^{20} c_i a_{n-i}$ or a list of implicit polynomial equations in variables $V = \{a_n, a_{n-1}, \ldots, a_{n-20}\}$.

```
1  N = [ ]
2  for o ∈ {1, 2, ..., 20} do
3  |   Construct observation matrix X from the first 2 · o + 10 elements of a_i,
   |    corresponding to variables in V = {a_n, a_{n−1}, ..., a_{n−o}}.
4  |   L = MoadeeB(X, mag = 50, len = 10, top_n = 10)
5  |   for p ∈ L do
6  |   |   if Validate(p, A) and p ∉ N then
7  |   |   |   N.append(p)
8  |   |   |   if IsLinear(p) then
9  |   |   |   |   return p
10 |   |   |   end
11 |   |   end
12 |   end
13 end
14 return N
```

The most notable change in the experimental setup for the core sequences (Algorithm B.2), not discussed in Section 4.2, is the attempt to convert implicit equations to explicit form (line 8). If successful and the explicit equation is correct (line 8), it is returned. Otherwise, the algorithm returns the list of implicit equations.

The need for explicit equations stems from two primary reasons. First, as discussed in Section 4.2, some implicit equations offer limited new information. Second, and more importantly, explicit forms are required for comparison with the transformer-based method of [34], described in Subsection 4.2.4.

We attempted to convert implicit equations to explicit form using the `solve` function

from the SymPy Python library [120], effectively solving for a_n. Specifically, we excluded solutions involving imaginary numbers. Furthermore, we limited the use of Cardano's formulas to degree 3 (setting `quartics=False`) to avoid complex piecewise solutions from SymPy. If no solution was found or the solutions were incorrect for the sequence elements, the equation was discarded.

Algorithm B.2: Evaluating *MoadeeB* on the *core* data set

Input: List A of the first 200 elements of a sequence a_i.

Output: An explicit equation of the form $a_n = q(n, a_{n-1}, \ldots, a_{n-10})$, where q is a rational expression, and a list of implicit polynomial equations.

$N = [\,]$
for $o \in \{0, 1, 2, \ldots, 10\}$ **do**
  Construct observation matrix X from the first $2 \cdot o + 10$ elements of a_i, corresponding to variables in $V = \{a_n, a_{n-1}, \ldots, a_{n-o}, n\}$.
  $L = \mathit{MoadeeB}(X, mag = 50, len = 20, top_n = 10)$
  for $p \in L$ **do**
    if $\mathit{Validate}(p, A)$ **and** $p \notin N$ **then**
      $N.\mathit{append}(p)$
      if $\mathit{ToExplicit}(p)$ returns a valid explicit equation e **then**
        return e, N
      end
    end
  end
end
return N

Appendix C

Equivalence of Linear Recurrences Through Generating Functions

We can determine whether two linear recurrences define the same integer sequence by comparing their generating functions: two sequences coincide if and only if their generating functions do. We therefore compute the generating functions associated with both recurrences and check whether they are equal. For linear recurrences, an explicit formula can be derived that expresses the corresponding generating function as a rational function. In the remainder of this section, we derive this formula.

Let (a_n) be an integer sequence defined by the linear recurrence relation

$$a_n = c_0 + \sum_{i=1}^{k} c_i a_{n-i}, n \geq k$$

together with the initial values $a_0, a_1, \ldots, a_{k-1}$. Note that $c_0, c_1, \ldots, c_k$ are integer coefficients, where $c_k \neq 0$. We refer to k as the *order* of the recurrence.

An example of an integer sequence with linear recurrence is the Fibonacci sequence (f_n), where $f_n = f_{n-1} + f_{n-2}$ with initial conditions $f_0 = f_1 = 1$. Note that the recurrence for the Fibonacci sequence is of order $k = 2$ with coefficients $c_0 = 0$, $c_1 = 1$, and $c_2 = 1$.

We derive the explicit formula for the generating function $G(x) = \sum_{n=0}^{\infty} a_n x^n$ of the sequence (a_n) in two steps:

1. deduce an equation involving the generating function $G(x)$ from the linear recurrence relation, and

2. solve the equation for $G(x)$ to get an explicit formula for the generating function.

C.1 Equation Involving the Generating Function

Given the linear recurrence, it is evident that

$$\sum_{n=k}^{\infty} \left(a_n - c_0 - \sum_{i=1}^{k} c_i a_{n-i} \right) x^n = 0.$$

We now examine the terms in the equation above. From:

$$\sum_{n=k}^{\infty} a_n x^n = G(x) - \sum_{n=0}^{k-1} a_n x^n,$$

$$\sum_{n=k}^{\infty} c_0 x^n = H(x) - \sum_{n=0}^{k-1} c_0 x^n,$$

$$\begin{aligned}\sum_{n=k}^{\infty} c_j a_{n-j} x^n &= c_j x^j \sum_{n=k}^{\infty} a_{n-j} x^{n-j} \\ &= c_j x^j \sum_{n=k-j}^{\infty} a_n x^n \\ &= c_j x^j \left(G(x) - \sum_{n=0}^{k-j-1} a_n x^n \right) \\ &= c_j x^j G(x) - \sum_{n=j}^{k-1} c_j a_{n-j} x^n,\end{aligned}$$

we derive:

$$\begin{aligned}\sum_{n=k}^{\infty} c_1 a_{n-1} x^n &= c_1 x G(x) - \sum_{n=1}^{k-1} c_1 a_{n-1} x^n, \\ \sum_{n=k}^{\infty} c_k a_{n-k} x^n &= c_k x^k G(x),\end{aligned}$$

where $H(x)$ denotes the generating function for the constant sequence (b_n), defined by $b_n = c_0$. Substituting these expressions into the original equation, we obtain

$$G(x) - p_0(x) - H(x) + \sum_{n=0}^{k-1} c_0 x^n - \sum_{i=1}^{k} \big(c_i x^i G(x) - p_i(x) \big) = 0,$$

where

$$\begin{aligned}p_0(x) &= \sum_{n=0}^{k-1} a_n x^n \\ p_i(x) &= \begin{cases} \sum_{n=i}^{k-1} c_i a_{n-i} x^n & \text{for } 1 \le i < k \\ 0 & \text{for } i = k. \end{cases}\end{aligned}$$

We now derive the explicit expression for the generating function $H(x)$ of the constant sequence (b_n), $b_n = c_0$. Note that

$$\begin{aligned}H(x) &= \sum_{n=0}^{\infty} c_0 x^n = c_0 + c_0 \sum_{n=1}^{\infty} x^n \\ xH(x) &= x \sum_{n=0}^{\infty} c_0 x^n = c_0 \sum_{n=0}^{\infty} x^{n+1} = c_0 \sum_{n=1}^{\infty} x^n.\end{aligned}$$

By subtracting the two equations above, we get $(1-x)H(x) = c_0$, which leads to

$$H(x) = \frac{c_0}{1-x}.$$

C.2 Solving the Equation

We now solve the equation for $G(x)$:

$$G(x) - p_0(x) - H(x) + \sum_{n=0}^{k-1} c_0 x^n - \sum_{i=1}^{k} \left(c_i x^i G(x) - p_i(x)\right) = 0.$$

By rearranging the terms, we get

$$\left(1 - \sum_{i=1}^{k} c_i x^i\right) G(x) = \frac{c_0}{1-x} - \sum_{n=0}^{k-1} c_0 x^n + p_0(x) - \sum_{i=1}^{k} p_i(x)$$

$$(1-x)\left(1 - \sum_{i=1}^{k} c_i x^i\right) G(x) = c_0 + (1-x)\left(p_0(x) - \sum_{n=0}^{k-1} c_0 x^n - \sum_{i=1}^{k} p_i(x)\right).$$

Finally,

$$G(x) = \frac{c_0 + (1-x)\left(p_0(x) - \sum_{n=0}^{k-1} c_0 x^n - \sum_{i=1}^{k} p_i(x)\right)}{(1-x)\left(1 - \sum_{i=1}^{k} c_i x^i\right)}$$

and, if we consider $p_k(x) = 0$, we get:

$$G(x) = \frac{c_0 + (1-x)\left(p_0(x) - \sum_{n=0}^{k-1} c_0 x^n - \sum_{i=1}^{k-1} p_i(x)\right)}{(1-x)\left(1 - \sum_{i=1}^{k} c_i x^i\right)}.$$

For linear recurrences with no free coefficient, i.e., with $c_0 = 0$, this simplifies to

$$G(x) = \frac{p_0(x) - \sum_{i=1}^{k-1} p_i(x)}{\left(1 - \sum_{i=1}^{k} c_i x^i\right)}.$$

C.3 Checking the Equivalence

To check whether two linear recurrences are equivalent, we derive the corresponding generating functions $G_1(x)$ and $G_2(x)$ and check whether $G_1(x) = G_2(x)$. Recall from the previous section that

$$G_1(x) = \frac{P_1(x)}{Q_1(x)}$$
$$G_2(x) = \frac{P_2(x)}{Q_2(x)}.$$

So, we have to check whether $P_1(x)Q_2(x) = Q_1(x)P_2(x)$. Note that the expressions on the left- and right-hand sides of the equation involve only operations on polynomials. Thus, only operations on polynomials are required to check whether two linear recurrence relations are equivalent.

The equivalence-checking algorithm is implemented in two files. The file `Polynomial.py` implements a class for operations on polynomials. The file `GenFunLinRec.py` implements a class for computing the generating function of a given linear recurrence and for checking equality between two generating functions, as explained above. The two files are available in our repository [117].

References

[1] The OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: http://oeis.org.

[2] J. Conway, R. Curtis, S. Norton, R. Parker, and R. Wilson, *Atlas of Finite Groups*. Oxford University Press, Eynsham, 1985.

[3] R. Wilson et al., *ATLAS of Finite Group Representations*, version 3, 2022. Accessed: Apr. 8, 2026. [Online]. Available: https://brauer.maths.qmul.ac.uk/Atlas/v3/.

[4] G. Brinkmann, K. Coolsaet, J. Goedgebeur, and H. Mélot, "House of Graphs: A database of interesting graphs," *Discrete Applied Mathematics*, vol. 161, no. 1, pp. 311–314, 2013. DOI: 10.1016/j.dam.2012.07.018.

[5] T. LMFDB Collaboration, *The L-functions and modular forms database*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://www.lmfdb.org.

[6] S. Welleck, J. Liu, X. Lu, H. Hajishirzi, and Y. Choi, "NATURALPROVER: Grounded Mathematical Proof Generation with Language Models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, New Orleans, LA, USA: Curran Associates, Inc., 2022, pp. 4913–4927. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/1fc548a8243ad06616eee731e0572927-Paper-Conference.pdf.

[7] J. M. Han, J. Rute, Y. Wu, E. Ayers, and S. Polu, "Proof Artifact Co-Training for Theorem Proving with Language Models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=rpxJc9j04U.

[8] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong, "Solving olympiad geometry without human demonstrations," *Nature*, vol. 625, no. 7995, pp. 476–482, Jan. 2024. DOI: 10.1038/s41586-023-06747-5.

[9] A. Z. Wagner, *Constructions in combinatorics via neural networks*, 2021. DOI: 10.48550/ARXIV.2104.14516.

[10] B. Romera-Paredes et al., "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, Dec. 2023. DOI: 10.1038/s41586-023-06924-6.

[11] A. Davies et al., "Advancing mathematics by guiding human intuition with AI," *Nature*, vol. 600, pp. 70–74, Dec. 2021. DOI: 10.1038/s41586-021-04086-x.

[12] Y.-H. He, K.-H. Lee, T. Oliver, and A. Pozdnyakov, "Murmurations of Elliptic Curves," *Experimental Mathematics*, pp. 1–13, Aug. 2024. DOI: 10.1080/10586458.2024.2382361.

[13] N. Zubrilina, "Murmurations," *Inventiones mathematicae*, vol. 241, no. 3, pp. 627–680, Sep. 2025. DOI: 10.1007/s00222-025-01347-8.

[14] W. Sawin and A. V. Sutherland, *Murmurations for elliptic curves ordered by height*, 2025. DOI: 10.48550/arXiv.2504.12295.

[15] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969. DOI: 10.1109/TIT.1969.1054260.

[16] B. Salvy and P. Zimmermann, "GFUN: a Maple package for the manipulation of generating and holonomic functions in one variable," *ACM Trans. Math. Softw.*, vol. 20, no. 2, pp. 163–177, Jun. 1994. DOI: 10.1145/178365.178368.

[17] M. Kauers, *Guess - A package for guessing multivariate recurrence equations, implemented for Mathematica 7*, version 0.59, Feb. 2022. Accessed: Apr. 8, 2026. [Online]. Available: http://www.kauers.de/software.html.

[18] M. Kauers, M. Jaroschek, and F. Johansson, "Ore Polynomials in Sage," in *Computer Algebra and Polynomials: Applications of Algebra and Number Theory*, J. Gutierrez, J. Schicho, and M. Weimann, Eds. Cham: Springer International Publishing, 2015, pp. 105–125. DOI: 10.1007/978-3-319-15081-9_6.

[19] E. Luzhnica and M. Kohlhase, "Formula Semantification and Automated Relation Finding in the On-Line Encyclopedia for Integer Sequences," in *In Proceedings of the International Congress on Mathematical Software*, vol. 9725, Paris, France: ICMS-2016, 2016, pp. 467–475. DOI: 10.1007/978-3-319-42432-3_60.

[20] G. Caporossi and P. Hansen, "Finding relations in polynomial time," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'99, Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 780–785.

[21] P. Hansen and G. Caporossi, "AutoGraphiX: An Automated System for Finding Conjectures in Graph Theory," *Electronic Notes in Discrete Mathematics*, vol. 5, pp. 158–161, 2000, 6th International Conference on Graph Theory. DOI: 10.1016/S1571-0653(05)80151-9.

[22] L. Todorovski, "Equation Discovery," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds., Boston, MA: Springer US, 2010, pp. 327–330. DOI: 10.1007/978-0-387-30164-8_258.

[23] J. Koza, M. Keane, and J. Rice, "Performance improvement of machine learning via automatic discovery of facilitating functions as applied to a problem of symbolic system identification," in *IEEE International Conference on Neural Networks*, 1993, 191–198 vol.1.

[24] M. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009. DOI: 10.1126/science.1165893.

[25] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016. DOI: 10.1073/pnas.1517384113.

[26] L. Todorovski and S. Džeroski, "Declarative Bias in Equation Discovery," in *Proceedings of the Fourteenth International Conference on Machine Learning*, ser. ICML '97, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 376–384. [Online]. Available: https://dl.acm.org/doi/10.5555/645526.657279.

[27] J. Brence, L. Todorovski, and S. Džeroski, "Probabilistic grammars for equation discovery," *Knowledge-Based Systems*, vol. 224, Apr. 2021. DOI: 10.1016/j.knosys.2021.107077.

[28] G. Martius and C. H. Lampert, *Extrapolation and learning equations*, 2016. DOI: 10.48550/arXiv.1610.02995.

[29] S. Sahoo, C. Lampert, and G. Martius, "Learning Equations for Extrapolation and Control," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 4442–4450. [Online]. Available: https://proceedings.mlr.press/v80/sahoo18a.html.

[30] S.-M. Udrescu and M. Tegmark, "AI Feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, no. 16, eaay2631, 2020. DOI: 10.1126/sciadv.aay2631.

[31] T. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. Faissol, and B. K. Petersen, "Symbolic Regression via Deep Reinforcement Learning Enhanced Genetic Programming Seeding," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 24 912–24 923.

[32] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar Variational Autoencoder," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, pp. 1945–1954.

[33] S. Mežnar, S. Džeroski, and L. Todorovski, "Efficient generator of mathematical expressions for symbolic regression," *Machine Learning*, 2023. DOI: 10.1007/s10994-023-06400-2.

[34] S. d'Ascoli, P.-A. Kamienny, G. Lample, and F. Charton, "Deep symbolic regression for recurrence prediction," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, Jul. 2022, pp. 4520–4536. [Online]. Available: https://proceedings.mlr.press/v162/d-ascoli22a.html.

[35] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton, "End-to-end Symbolic Regression with Transformers," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 10 269–10 281. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/42eb37cdbefd7abae0835f4b67548c39-Paper-Conference.pdf.

[36] M. Valipour, B. You, M. Panju, and A. Ghodsi, *SymbolicGPT: A Generative Transformer Model for Symbolic Regression*, 2022. DOI: 10.48550/arXiv.2106.14131.

[37] H. M. Möller and B. Buchberger, "The construction of multivariate polynomials with preassigned zeros," in *Computer Algebra*, J. Calmet, Ed., Berlin: Springer Berlin Heidelberg, 1982, pp. 24–31. DOI: 10.1007/3-540-11607-9_3.

[38] T. Gauthier and J. Urban, "Learning Program Synthesis for Integer Sequences from Scratch," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 6, pp. 7670–7677, Jun. 2023. DOI: 10.1609/aaai.v37i6.25930.

[39] R. McCasland and A. Bundy, "MATHsAiD: A Mathematical Theorem Discovery Tool," English, in *Symbolic and Numeric Algorithms for Scientific Computing, 2006. SYNASC '06. Eighth International Symposium on*, United States: Institute of Electrical and Electronics Engineers, 2006, pp. 17–22. DOI: 10.1109/SYNASC.2006.51.

[40] J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban, "Hammering towards QED," *Journal of Formalized Reasoning*, vol. 9, no. 1, pp. 101–148, 2016. DOI: 10.6092/issn.1972-5787/4593.

[41] G. Irving, C. Szegedy, A. A. Alemi, N. Een, F. Chollet, and J. Urban, "DeepMath - deep sequence models for premise selection," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016, pp. 2243–2253. [Online]. Available: https://dl.acm.org/doi/10.5555/3157096.3157347.

[42] A. Paliwal, S. Loos, M. Rabe, K. Bansal, and C. Szegedy, "Graph Representations for Higher-Order Logic and Theorem Proving," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, pp. 2967–2974, Apr. 2020. DOI: 10.1609/aaai.v34i03.5689.

[43] A. Bauer, M. Petković, and L. Todorovski, "MLFMF: data sets for machine learning for mathematical formalization," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23, New Orleans, LA, USA: Curran Associates Inc., 2024, pp. 50 730–50 741. [Online]. Available: https://dl.acm.org/doi/10.5555/3666122.3668329.

[44] M. Wu, M. Norrish, C. Walder, and A. Dezfouli, "TacticZero: Learning to Prove Theorems from Scratch with Deep Reinforcement Learning," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, ser. NIPS '21, Red Hook, NY, USA: Curran Associates Inc., 2021.

[45] A. Sanchez-Stern, Y. Alhessi, L. Saul, and S. Lerner, "Generating correctness proofs with neural networks," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL 2020, London, UK: Association for Computing Machinery, 2020, pp. 1–10. DOI: 10.1145/3394450.3397466.

[46] D. B. Lenat, "The ubiquity of discovery," *Artificial Intelligence*, vol. 9, no. 3, pp. 257–285, 1977. DOI: 10.1016/0004-3702(77)90024-8.

[47] M. Märtens, F. Kuipers, and P. Van Mieghem, "Symbolic Regression on Network Properties," in *Genetic Programming*, J. McDermott, M. Castelli, L. Sekanina, E. Haasdijk, and P. García-Sánchez, Eds., Cham: Springer International Publishing, 2017, pp. 131–146.

[48] G. Raayoni et al., "Generating conjectures on fundamental constants with the Ramanujan Machine," *Nature*, vol. 590, pp. 67–73, Feb. 2021. DOI: 10.1038/s41586-021-03229-4.

[49] M. Kreuzer and H. Skarke, "Complete classification of reflexive polyhedra in four dimensions," *Advances in Theoretical and Mathematical Physics*, vol. 4, no. 6, pp. 1209–1230, Oct. 2000. DOI: 10.4310/ATMP.2000.v4.n6.a2.

[50] Y.-H. He and S.-J. Lee, "Distinguishing elliptic fibrations with AI," *Physics Letters B*, vol. 798, p. 134 889, Nov. 2019. DOI: 10.1016/j.physletb.2019.134889.

[51] Y.-H. He and A. Lukas, "Machine learning Calabi-Yau four-folds," *Physics Letters B*, vol. 815, p. 136 139, Apr. 2021. DOI: 10.1016/j.physletb.2021.136139.

[52] T. Coates, A. M. Kasprzyk, and S. Veneziale, "Machine learning the dimension of a Fano variety," *Nature Communications*, vol. 14, pp. 2041–1723, 1 Sep. 2023. DOI: 10.1038/s41467-023-41157-1.

[53] T. Coates, A. Kasprzyk, and S. Veneziale, "Machine learning detects terminal singularities," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36, Curran Associates, Inc., 2023, pp. 67 183–67 194. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/d453490ada2b1991852f053fbd213a6a-Paper-Conference.pdf.

[54] S. Colton, "Computational Discovery in Pure Mathematics," in *Computational Discovery of Scientific Knowledge: Introduction, Techniques, and Applications in Environmental and Life Sciences*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ch. Computational Discovery in Pure Mathematics, pp. 175–201. DOI: 10.1007/978-3-540-73920-3_9.

[55] S. C. Billey and B. E. Tenner, "Fingerprint Databases for Theorems," *Notices of the American Mathematical Society*, vol. 60, no. 08, p. 1034, Sep. 2013. DOI: 10.1090/noti1029.

[56] W. R. Inc., *Wolfram Mathematica*, version 14.3, Champaign, IL, USA, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://www.wolfram.com/mathematica.

[57] S. Džeroski and L. Todorovski, Eds., *Computational Discovery of Scientific knowledge* (Lecture notes in computer science), en. Berlin, Germany: Springer, Aug. 2007. DOI: 10.1007/978-3-540-73920-3.

[58] P. Langley, "Data-Driven Discovery of Physical Laws," *Cognitive Science*, vol. 5, no. 1, pp. 31–54, 1981. DOI: 10.1111/j.1551-6708.1981.tb00869.x.

[59] S. Džeroski and L. Todorovski, "Discovering Dynamics," in *Machine Learning Proceedings 1993*, San Francisco (CA): Morgan Kaufmann, 1993, pp. 97–103. DOI: 10.1016/B978-1-55860-307-3.50019-8.

[60] S. Džeroski and I. Petrovski, "Discovering dynamics with genetic programming," in *Machine Learning: ECML-94*, F. Bergadano and L. De Raedt, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 347–350. DOI: 10.1007/3-540-57868-4_70.

[61] M. Cranmer, *Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl*, 2023. DOI: 10.48550/arXiv.2305.01582.

[62] W. Bridewell, P. Langley, L. Todorovski, and S. Džeroski, "Inductive process modeling," *Machine Learning*, vol. 71, pp. 1–32, Apr. 2008. DOI: 10.1007/s10994-007-5042-6.

[63] L. Todorovski, "Inductive Process Modeling," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 537–537. DOI: 10.1007/978-0-387-30164-8_397.

[64] J. Tanevski, L. Todorovski, and S. Džeroski, "Combinatorial search for selecting the structure of models of dynamical systems with equation discovery," *Engineering Applications of Artificial Intelligence*, vol. 89, p. 103 423, 2020. DOI: 10.1016/j.engappai.2019.103423.

[65] D. Čerepnalkoski, "Process-Based Models of Dynamical Systems: Representation and Induction," Ph.D. dissertation, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, 2013.

[66] P. Shojaee, K. Meidani, S. Gupta, A. B. Farimani, and C. K. Reddy, "LLM-SR: Scientific Equation Discovery via Programming with Large Language Models," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=m2nmp8P5in.

[67] P. Breiding, S. Kališnik, B. Sturmfels, and M. Weinstein, "Learning algebraic varieties from samples," *Revista Matemática Complutense*, vol. 31, no. 3, pp. 545–593, Sep. 2018. DOI: 10.1007/s13163-018-0273-6.

[68] B. Buchberger, "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal," German, PhD thesis introducing Gröbner bases, with an English translation published online in 2006, Ph.D. dissertation, University of Innsbruck, 1965. DOI: 10.1016/j.jsc.2005.09.007.

[69] M. G. Marinari, H. M. Möller, and T. Mora, "Gröbner bases of ideals defined by functionals with an application to ideals of projective points," *Applicable Algebra in Engineering, Communication and Computing*, vol. 4, no. 2, pp. 103–145, Jun. 1993. DOI: 10.1007/BF01386834.

[70] J. Abbott, A. Bigatti, M. Kreuzer, and L. Robbiano, "Computing Ideals of Points," *Journal of Symbolic Computation*, vol. 30, no. 4, pp. 341–356, 2000. DOI: 10.1006/jsco.2000.0411.

[71] J. B. Farr and S. Gao, "Computing Gröbner Bases for Vanishing Ideals of Finite Sets of Points," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, M. P. C. Fossorier, H. Imai, S. Lin, and A. Poli, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 118–127. DOI: 10.1007/11617983_11.

[72] W. Just and B. Stigler, *Efficiently computing Gröbner bases of ideals of points*, 2007. DOI: 10.48550/arXiv.0711.3475.

[73] D. Heldt, M. Kreuzer, S. Pokutta, and H. Poulisse, "Approximate computation of zero-dimensional polynomial ideals," *Journal of Symbolic Computation*, vol. 44, no. 11, pp. 1566–1591, 2009, In Memoriam Karin Gatermann. DOI: 10.1016/j.jsc.2008.11.010.

[74] R. Livni, D. Lehavi, S. Schein, H. Nachliely, S. Shalev-Shwartz, and A. Globerson, "Vanishing Component Analysis," in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, Jun. 2013, pp. 597–605. [Online]. Available: https://proceedings.mlr.press/v28/livni13.html.

[75] H. Kera and Y. Hasegawa, "Monomial-agnostic computation of vanishing ideals," *Journal of Computational Algebra*, vol. 11, p. 100 022, 2024. DOI: 10.1016/j.jaca.2024.100022.

[76] A. Karimov, E. G. Nepomuceno, A. Tutueva, and D. Butusov, "Algebraic Method for the Reconstruction of Partially Observed Nonlinear Systems Using Differential and Integral Embedding," *Mathematics*, vol. 8, no. 2, 2020. DOI: 10.3390/math8020300.

[77] H. Kera and Y. Hasegawa, "Noise-tolerant algebraic method for reconstruction of nonlinear dynamical systems," *Nonlinear Dynamics*, vol. 85, no. 1, pp. 675–692, Jul. 2016. DOI: 10.1007/s11071-016-2715-3.

[78] K. Kaheman, J. N. Kutz, and S. L. Brunton, "SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, no. 2242, 2020. DOI: 10.1098/rspa.2020.0279.

[79] P. Breiding, K. Kohn, and B. Sturmfels, *Metric Algebraic Geometry*, en, 2024th ed. Basel, Switzerland: Birkhäuser Cham, Feb. 2024. DOI: 10.1007/978-3-031-51462-3.

[80] C. Prony, "Essai expérimental et analytique sur les lois de la dilabilité des fluides élastiques, et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alkool, à différentes températures.," *J. de l' École polytechnique*, vol. 2, pp. 24–77, 1795.

[81] T. Sauer, "Prony's method: an old trick for new problems," en, *Snapshots of modern mathematics from Oberwolfach;2018*, p. 04, 2018. DOI: 10.14760/SNAP-2018-004-EN.

[82] T. Sauer, "Prony's method in several variables: Symbolic solutions by universal interpolation," *Journal of Symbolic Computation*, vol. 84, pp. 95–112, 2018. DOI: 10.1016/j.jsc.2017.03.006.

[83] C. Josz, J. B. Lasserre, and B. Mourrain, "Sparse polynomial interpolation: sparse recovery, super-resolution, or Prony?" *Advances in Computational Mathematics*, vol. 45, no. 3, pp. 1401–1437, Jun. 2019. DOI: 10.1007/s10444-019-09672-2.

[84] S. M. Hietpas and D. A. Pierre, "System Identification Using Prony Methods For Digital Control Systems," *IFAC Proceedings Volumes*, vol. 27, no. 8, pp. 903–908, 1994, IFAC Symposium on System Identification (SYSID'94), Copenhagen, Denmark, 4-6 July. DOI: 10.1016/S1474-6670(17)47824-4.

[85] M. Ben-Or and P. Tiwari, "A deterministic algorithm for sparse multivariate polynomial interpolation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 301–309. DOI: 10.1145/62212.62241.

[86] S. Sakata, "Finding a minimal set of linear recurring relations capable of generating a given finite two-dimensional array," *Journal of Symbolic Computation*, vol. 5, no. 3, pp. 321–337, 1988. DOI: 10.1016/S0747-7171(88)80033-6.

[87] W. F. Lunnon, "The Number-Wall Algorithm: an LFSR Cookbook," *Journal of Integer Sequences*, vol. 4, no. 01.1.1, 1 Oct. 2001. [Online]. Available: https://cs.uwaterloo.ca/journals/JIS/VOL4/LUNNON/numbwall10.html.

[88] F. Lunnon, "The Pagoda Sequence: a Ramble through Linear Complexity, Number Walls, D0L Sequences, Finite State Automata, and Aperiodic Tilings," in *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008*, T. Neary, D. Woods, T. Seda, and N. Murphy, Eds., ser. Electronic Proceedings in Theoretical Computer Science, vol. 1, Cork, Ireland: Open Publishing Association, 2009, pp. 130–148. DOI: 10.4204/EPTCS.1.13.

[89] F. Adiceam, E. Nesharim, and F. Lunnon, "On the t-adic Littlewood conjecture," *Duke Mathematical Journal*, vol. 170, no. 10, pp. 2371–2419, 2021. DOI: 10.1215/00127094-2020-0077.

[90] F. Bergeron and S. Plouffe, "Computing the Generating Function of a Series Given Its First Few Terms," English, *Exp. Math.*, vol. 1, no. 4, pp. 307–312, 1992. DOI: 10.1080/10586458.1992.10504563.

[91] B. Beckermann and G. Labahn, "Recursiveness in matrix rational interpolation problems," *Journal of Computational and Applied Mathematics*, vol. 77, no. 1, pp. 5–34, 1997. DOI: 10.1016/S0377-0427(96)00120-3.

[92] W. Hebisch and M. Rubey, "Extended rate, more GFUN," *Journal of Symbolic Computation*, vol. 46, no. 8, pp. 889–903, 2011. DOI: 10.1016/j.jsc.2011.01.004.

[93] C. Krattenthaler, *RATE - A Mathematica guessing machine*. Accessed: Apr. 8, 2026. [Online]. Available: https://www.mat.univie.ac.at/~kratt/rate/rate.html.

[94] J. Berthomieu and M. Safey El Din, "Guessing Gröbner bases of structured ideals of relations of sequences," *Journal of Symbolic Computation*, vol. 111, pp. 1–26, 2022. DOI: 10.1016/j.jsc.2021.11.001.

[95] K. M. G. Havas B.S. Majewski, "Extended GCD and Hermite Normal Form Algorithms via Lattice Basis Reduction," *Experimental Mathematics*, vol. 7, pp. 125–136, 1998. DOI: 10.1080/10586458.1998.10504362.

[96] S. Kopparty and A. Bhrushundi. "Lecture 3: Finding Integer Solutions to Systems of Linear Equations." Algorithmic Number Theory, Fall 2014, Rutgers University. [Online]. Available: https://sites.math.rutgers.edu/~sk1233/courses/ANT-F14/lec3.pdf.

[97] T. G. Close, *Diophantine: A Python package for finding small solutions of systems of Diophantine (integer algebra) equations*, version 0.2.0, Nov. 2017. Accessed: Apr. 8, 2026. [Online]. Available: https://pypi.org/project/Diophantine/.

[98] W. Van Der Kallen, "Complexity of the Havas, Majewski, Matthews LLL Hermite Normal Form Algorithm," *J. Symb. Comput.*, vol. 30, no. 3, pp. 329–337, Sep. 2000. DOI: 10.1006/jsco.2000.0374.

[99] S. D. Galbraith, "Lattice basis reduction," in *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012, pp. 347–365.

[100] D. Micciancio and B. Warinschi, "A linear space algorithm for computing the Hermite Normal Form," in *International Symposium on Symbolic and Algebraic Computation*, B. Mourrain, Ed., ser. ISSAC '01, ACM, London, Ontario, Canada: Association for Computing Machinery, 2001, pp. 231–236. DOI: 10.1145/384101.384133.

[101] The OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences Wiki, Index to OEIS: Section Rec*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://oeis.org/wiki/Index_to_OEIS:_Section_Rec.

[102] The OEIS Foundation Inc., *The On-Line Encyclopedia of Integer Sequences Wiki, Index to OEIS: Section Cor*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://oeis.org/wiki/Index_to_OEIS:_Section_Cor.

[103] B. Gec, *The databases of linear-recursive and "core" sequences*, Sep. 2024. DOI: 10.5281/zenodo.13767012.

[104] B. Gec, *B0Gec/Diofantos: Diophantos first paper release for Zenodo*, Sep. 2024. DOI: 10.5281/zenodo.13692310.

[105] A. A. Kaptanoglu et al., "PySINDy: A comprehensive Python package for robust sparse system identification [software]," *Journal of Open Source Software*, vol. 7, no. 69, p. 3994, 2022. DOI: 10.21105/joss.03994.

[106] OEIS Foundation Inc., *Entry A078475 in The On-Line Encyclopedia of Integer Sequences*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://oeis.org/A078475.

[107] OEIS Foundation Inc., *Entry A000330 in The On-Line Encyclopedia of Integer Sequences*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://oeis.org/A000330.

[108] I. R. Shafarevich, *Basic Algebraic Geometry 1*, Third. Berlin: Springer, 2013. DOI: 10.1007/978-3-642-37956-7.

[109] A. Altman and S. Kleiman, *A term of Commutative Algebra*, en. Cambridge, MA, USA: Worldwide Center of Mathematics, Apr. 2021. Accessed: Apr. 8, 2026. [Online]. Available: https://hdl.handle.net/1721.1/116075.2.

[110] A. Gathmann, *Algebraic Geometry*, Class notes for Algebraic Geometry Master course, Kaiserslautern, Jul. 2026. Accessed: Sep. 7, 2026. [Online]. Available: https://agag-gathmann.math.rptu.de/class/alggeom-2021/alggeom-2021.pdf.

[111] E. R. Chang, "Gröbner Bases and Ideals of Points," English, M.S. thesis, Virginia Commonwealth University, 2006. DOI: 10.25772/CSW9-YE53.

[112] The Sage Developers, *SageMath, the Sage Mathematics Software System*, version 10.7, Aug. 2025. DOI: 10.5281/zenodo.14600115.

[113] J. Abbott, A. M. Bigatti, and L. Robbiano, *CoCoA: A system for doing Computations in Commutative Algebra*, version 5.4.1p, Mar. 2023. Accessed: Apr. 8, 2026. [Online]. Available: https://sites.google.com/view/cocoa-cocoalib.

[114] OEIS Foundation Inc., *Entry A001699 in The On-Line Encyclopedia of Integer Sequences*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://oeis.org/A001699.

[115] OEIS Foundation Inc., *Entry A002658 in The On-Line Encyclopedia of Integer Sequences*, 2026. Accessed: Apr. 8, 2026. [Online]. Available: https://oeis.org/A002658.

[116] A. R. Forsyth, "Theory of Differential Equations VI," in *Theory of Differential Equations VI*. Cambridge: Cambridge University Press, 1906, ch. XIX.

[117] B. Gec, *MoadeeB - first (pre-)release*, Feb. 2025. DOI: 10.5281/zenodo.18391903.

[118] B. Gec, *Development branch for MoadeeB and Diofantos [Code, Data]*, version 48fb7b9, Apr. 2026. Accessed: Apr. 17, 2026. [Online]. Available: https://github.com/B0Gec/Diofantos.

[119] J. Brence, "Probabilistic grammar-based equation discovery," English, Ph.D. dissertation, Jožef Stefan International Postgraduate School, Apr. 2024. [Online]. Available: https://kt.ijs.si/wp-content/uploads/2025/07/Thesis_Brence_final.pdf.

[120] A. Meurer et al., "SymPy: symbolic computing in Python," *PeerJ Computer Science*, vol. 3, e103, Jan. 2017. DOI: 10.7717/peerj-cs.103.

[121] G. Tsoukalas et al., *Advancing Mathematics Research with AI-Driven Formal Proof Search*, 2026. arXiv: 2605.22763 [cs.AI].

[122] C. Cornelio et al., "Combining data and theory for derivable scientific discovery with AI-Descartes," *Nature Communications*, vol. 14, no. 1, p. 1777, Apr. 2023. DOI: 10.1038/s41467-023-37236-y.

[123] Y. Zhu, K. Sun, Z. Luo, and L. Wang, "Progressive self-learning for domain adaptation on symbolic regression of integer sequences," in *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'25/IAAI'25/EAAI'25, AAAI Press, 2025. DOI: 10.1609/aaai.v39i2.32162.

[124] C. Livingston and A. H. Moore, *KnotInfo: Table of Knot Invariants*, Apr. 2026. Accessed: Apr. 23, 2026. [Online]. Available: https://knotinfo.org.

Bibliography

Publications Related to the Thesis

Journal Articles

B. Gec, S. Džeroski, and L. Todorovski, "Discovery of Exact Equations for Integer Sequences," *Mathematics*, vol. 12, no. 23, 2024. DOI: 10.3390/math12233745.

B. Gec, S. Džeroski, and L. Todorovski, "Discovery of Exact Equations via Computing the Gröbner Basis," *Machine Learning*, vol. 115, no. 8, p. 185, Jul. 2026. DOI: 10.1007/s10994-026-07112-z.

Other Publications

Journal Articles

N. Omejc, B. Gec, J. Brence, L. Todorovski, and S. Džeroski, "Probabilistic grammars for modeling dynamical systems from coarse, noisy, and partial data," *Machine Learning*, vol. 113, no. 10, pp. 7689–7721, Oct. 2024. DOI: 10.1007/s10994-024-06522-1.

Conference Paper

B. Gec, N. Omejc, J. Brence, S. Džeroski, and L. Todorovski, "Discovery of Differential Equations Using Probabilistic Grammars," in *Discovery Science*, P. Pascal and D. Ienco, Eds., Cham: Springer Nature Switzerland, 2022, pp. 22–31.

Biography

Boštjan Gec was born on 23. 4. 1994 in Šempeter pri Gorici. He finished his primary education at "OŠ Šturje" in Ajdovščina and his secondary education at "Škofijska gimnazija Vipava". In 2013 he began his study of Mathematics at the Faculty for mathematics and physics (FMF), University of Ljubljana. After completing his bachelor's degree in 2018, Boštjan Gec continued his studies of Mathematics at FMF at the master's level. During the additional third year of master's studies, he was intensively involved in research work in the equation discovery group at the Department of knowledge technologies at Jožef Stefan Institute (JSI) under the supervision of prof. dr. Ljupčo Todorovski. The research culminated with the defense of his Master's thesis, titled "Equation discovery for integer sequences with probabilistic grammars". In 2021 he enrolled in the PhD study programme Information and communication technologies at the Jožef Stefan International Postgraduate School and started working as a research assistant at the Department of knowledge technologies at JSI under the supervision of prof. dr. Ljupčo Todorovski and prof. dr. Sašo Džeroski. His research interests lie in the area of machine learning, especially its use in mathematics, with a focus on methods of equation discovery. He has published several scientific papers, presented his work at international conferences and taken part in summer schools on related topics.