mathematics

Article

Discovery of Exact Equations for Integer Sequences

Boštjan Gec ^{1,2,*}, Sašo Džeroski ^{1} and Ljupčo Todorovski ^{1,3}

1 Department of Knowledge Technologies, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia; saso.dzeroski@ijs.si (S.D.); ljupco.todorovski@fmf.uni-lj.si (L.T.)
2 Jožef Stefan International Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia
3 Faculty of Mathematics and Physics, University of Ljubljana, Jadranska 19, 1000 Ljubljana, Slovenia
* Correspondence: bostjan.gec@ijs.si

Abstract: Equation discovery, also known as symbolic regression, is the field of machine learning that studies algorithms for discovering quantitative laws, expressed as closed-form equations or formulas, in collections of observed data. The latter is expected to come from measurements of physical systems and, therefore, noisy, moving the focus of equation discovery algorithms towards discovering approximate equations. These loosely match the noisy observed data, rendering them inappropriate for applications in mathematics. In this article, we introduce *Diofantos*, an algorithm for discovering equations in the ring of integers that exactly match the training data. *Diofantos* is based on a reformulation of the equation discovery task into the task of solving linear Diophantine equations. We empirically evaluate the performance of *Diofantos* on reconstructing known equations for more than 27,000 sequences from the online encyclopedia of integer sequences, OEIS. *Diofantos* successfully reconstructs more than 90% of these equations and clearly outperforms SINDy, a state-of-the-art method for discovering approximate equations, that achieves a reconstruction rate of less than 70%.

Keywords: machine learning; equation discovery; symbolic regression; Diophantine equations; online encyclopedia of integer sequences (OEIS)

MSC: 68T05; 68U99; 11Y50

check for updates

Citation: Gec, B.; Džeroski, S.; Todorovski, L. Discovery of Exact Equations for Integer Sequences. *Mathematics* **2024**, *12*, 3745. https://doi.org/10.3390/math12233745

Academic Editor: Faheim Sufi

Received: 3 October 2024
Revised: 16 November 2024
Accepted: 24 November 2024
Published: 28 November 2024

1. Introduction

Modeling and aiding discovery processes in mathematics with artificial intelligence and machine learning has a long history. The early efforts towards automated mathematics [1] have been based on a collection of hand-crafted heuristics for the discovery of new mathematical concepts and conjectures related to them. More recently, there has been a growing interest in using machine learning to address the task of premise selection [2–7], i.e., training models for the recommendation of theorems that are useful for proving a given statement in proof assistants, which verify the formal correctness of mathematical proofs and constructions. Also, machine learning has been used for discovering patterns and predictive models in databases of mathematical objects that lead to novel conjectures proved by mathematicians [8,9]. The patterns discovered in databases of mathematical objects often take the form of closed-form equations relating to the properties of the objects. To this end, machine learning methods for equation discovery, also known as symbolic regression, can be used. Given a data set of observations of a collection of numerical variables, equation discovery methods can automate learning quantitative laws, expressed as closed-forms equations, among the observed variables. The history of equation discovery is long and ranges from early efforts based on data-driven heuristics [10] through the use of evolutionary approaches [11] and search techniques [12,13], to deep learning [14,15] and sparse regression [16]. The equation discovery algorithms have been widely used for modelling real-world systems from data in various scientific and engineering fields, such as physics [11] or earth sciences [13]. Note, however, that the utility of these methods for discovering relations in pure mathematics is limited due to their focus on approximate equations that loosely fit the observed data.

In this article, we aim to develop a new approach to equation discovery that guarantees the exact match of the equations to the observed data in the ring of integers. Our approach is similar to the one of SINDy [16], which is based on using sparse linear regression on an extensive set of non-linear transformations of the originally observed variables. SINDy, in turn, is very similar to Lagrange [17], with the most significant difference being the use of regularized versus ordinary linear regression. Both are in contrast with the most common approach to equation discovery, which decomposes the task of equation discovery into the two interleaved subtasks of structure identification (i.e., finding the proper equation structure with unknown values of the constant parameters) and parameter estimation (i.e., finding the values of the constant parameters that minimize the discrepancy between the observed data and data obtained with using/simulating the equation). Common equation discovery approaches would address the structure identification subtask using various techniques for combinatorial optimization, ranging from evolutionary approaches [11] to greedy search [18]. Various methods for numerical optimization are used to tackle the parameter estimation subtask.

SINDy efficiently addresses both subtasks simultaneously using sparse regression. The first phase applies multiplication and a set of other user-specified functions to the initial set of observed variables to obtain an extensive collection of non-linear, higher-order terms. The second phase uses sparse linear regression on the extended set of terms, including the observed variables and the non-linear terms introduced in the first phase, to find an appropriate parsimonious equation that fits the observed data well. This approach is computationally very efficient but limited to the discovery of equations that are linear with respect to the constant parameters. Despite this limitation, it has been proven useful in various scenarios of discovering approximate equations. Our system, *Diofantos*, follows the SINDy approach in the first phase and introduces higher-order terms to be linearly combined into expressions on the right-hand side of the equations. In the second phase, *Diofantos* employs a solver of linear Diophantine equations [19] to find constant parameters that provide an exact match to the observed data.

We will empirically evaluate the performance of *Diofantos* on discovering recursive equations for two sets of integer sequences from the OEIS, the online encyclopedia of integer sequences [20]. The first set consists of 27,236 sequences with known linear recursive equations of various orders. The second set includes the 164 core sequences from the OEIS. We measure the performance of equation discovery as the number of successfully reconstructed known equations and compare the performance of *Diofantos* with the one of SINDy. We conjecture that *Diofantos*, due to its ability to discover exact equations, will outperform SINDy on both sets of integer sequences.

The rest of the manuscript is organized as follows. Section 2 reviews the related work on equation discovery, symbolic regression, and using artificial intelligence and machine learning for discovery in mathematics. Section 3 introduces the task of discovering exact equations and the *Diofantos* algorithm. Section 4 reports and discusses the results of the empirical evaluation of *Diofantos* on the two sets of sequences from OEIS. Finally, Section 5 summarizes the contributions and outlines directions for further research.

2. Related Work

The use of computational discovery in mathematics has recently become a very active research field. We observe significant progress in automated theorem proving, reviewed in the first subsection below. In contrast, equation discovery techniques, reviewed in the second subsection, have yet to make important contributions to mathematics due to their focus on approximate equations that lack the rigor and precision required for mathematical proofs and formalizations.

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 theory proving (ATP), as mathematicians' affinity towards theorems, lemmas, corollaries, etc., makes it a natural fit for automation [21]. ATP has seen significant advancements through methods like "hammering", which leverages computational power to supply relevant information to formal provers and complete proofs [2–7]. Recent work by [5,7,22] exemplifies this approach by employing large language models (LLMs) similar to GPT-3 to predict and complete proofs effectively. An empirical study by [22] 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 [1]. 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.

AutoGraphiX, developed by [23], is a computational system that supports discoveries in graph theory. Its capabilities include identifying extremal graphs based on specific graph invariants, locating graphs satisfying given constraints, and even refuting or proposing conjectures related to graph invariants and suggesting potential proofs.

In [24], the authors utilize reinforcement learning (RL) 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. RL guides the probability distribution of training examples for a deep neural network, favoring those with higher scores that may indicate counterexamples. Moreover, ref. [8] employed machine learning for predictive modeling and estimating the predictive importance of features to identify novel insights in pure mathematics. Their efforts resulted in two significant discoveries: an inequality of invariants of knots in the field of knot theory and a finding related to invariants of graphs in the context of graph theory.

In addition to addressing OEIS, central to our paper, ref. [25] reviews various experimental mathematics methods and mathematical data resources pertinent to the topic. His approach, HR, generates conjectures based on examples from a given mathematical theory. HR has successfully identified theorems in algebra and group theory, particularly within the context of anti-associative algebras, quasigroups, and anti-Abelian groups. Furthermore, it has been applied to the invention of numerous sequences worthy of inclusion in OEIS. HR has also been used to discover conjectures about these sequences and uncover empirical relationships among them, such as identifying subsequences and sequences without shared terms.

In [26], the authors 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, ref. [27] carries out automated discovery on the OEIS database. The authors explore relations between integer sequences using the closed forms of their generating functions. If the closed forms of two sequences can be calculated, they can be compared. Through this method, they discovered more than 60,000 new relations.

The authors of [28] 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 (TNNs) 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 yielded 27,987 correct computer programs within the OEIS, a repository of 351,663 integer sequences at the time of their investigation.

2.2. Discovery of Equations from Data

The pioneer method for equation discovery, BACON [10], 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. The approaches can be classified into four categories based on the methodologies and principles used. The first category, including Lagrange [17] and SINDy [16], employs linear regression to discover linear relationships among higher-order terms compiled from the observed variables. SINDy utilizes sparse regression techniques for parameter fitting, effectively managing model complexity despite the high number of observed variables and the higher-order terms.

The second category of methods employs evolutionary computing [29], and also includes GPDD [30], Eureka [11], CSR [31], and PySR [32]. 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 algorithms.

The third category employs background knowledge from the application domain to constrain the search space for candidate equations. Lagrange [12] and ProGED [33] leverage context-free grammars (CFGs) to constrain the search space. ProGED introduces soft constraints by employing a probabilistic version (PCFG) and performing Monte Carlo sampling to navigate the potentially vast search space efficiently. Process-based modeling [13,18] 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.

In this category, we can also cluster two approaches to discovering equations in pure mathematics. The Ramanujan Machine [34] algorithmically discovered new equations and conjectures related to Polynomial Continued Fractions (PCFs). 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.

Moreover, ref. [35] focuses on equation discovery's parameter estimation side. It introduces a likelihood-type cost function to estimate model parameters for differential equations, addressing incomplete and noisy data with unknown uncertainties. While balancing these uncertainties effectively, it accommodates missing information and demonstrates superior performance compared to current model discovery and calibration regression methods.

Finally, the fourth equation discovery method category employs neural networks and deep learning. EQL [36] 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 [37] exemplifies a physics-inspired approach, leveraging symmetries, separability, compositionality, and other simplifying properties inherent in physics. Different approaches, such as DSO [14], combine genetic programming and reinforcement learning to achieve state-of-the-art results. Some methods create an embedding of the search space of candidate equations using variational autoencoders (GVAE by [38] and EDHiE by [15]), enabling the discovery of equations with an arbitrary optimization algorithm. Furthermore, ref. [39] and [40] train transformers with data sets and the underlying equations as input–output pairs. The paper by [39] represents a nearly unique intersection of equation discovery and automated computational discovery in pure mathematics. This approach, based on the paper by [41], discovers the equations governing integer sequences of the OEIS, a task closely related to the focus of the present article. It outperforms two built-in functions of the well-known Mathematica software [42].

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.

3. Discovery of Exact Equations with *Diofantos*

This section defines the general task of discovering exact equations from data and its specific instance of discovering recursive equations for integer sequences. Next, it introduces *Diofantos*, an algorithm for discovering exact equations using a solver of linear Diophantine equations.

3.1. Task

The task of discovering exact integer equations from data can be formalized as follows:

Given observations of the values of the set of integer 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 $e\colon \mathbb{Z}^p \to \mathbb{Z}$ 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*.

An example of this task is provided in Table 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 [11,12].

Table 1. A simple example of discovering an exact integer equation from data: given the data table of observations of three variables $V = \{x_1, x_2, y\}$, the task is to find an equation of the form $y = e(x_1, x_2)$ that can be used to calculate the value of the target variable y given the values of the variables in $V \setminus \{y\}$.

Given			**Find**
x_1	x_2	y	
−2	1	0	
−1	2	3	$y = x_1 + 2x_2$
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, *find* a recursive formula that can be used to calculate the sequence elements. This task can be transformed or reformulated into the general task of discovering exact integer equations introduced above. In the next two paragraphs, we will introduce the transformation and illustrate it with an example of discovering the 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 2 illustrates the simple transformation of the sequence of the first eight Fibonacci numbers into a five-row data table for the discovery of recursive equations of order three ($o = 3$). 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 2. Transformation of the first eight Fibonacci numbers (bottom) into a data table of five observations of five variables for discovering a recursive equation of order three (top).

n	a_{n-3}	a_{n-2}	a_{n-1}	a_n
0	0	1	1	2
1	1	1	2	3
2	1	2	3	5
3	2	3	5	8
4	3	5	8	13

The sequence of the first eight Fibonacci numbers, 0, 1, 1, 2, 3, 5, 8, 13.

3.2. The Diofantos Algorithm

Based on the definition of the task of discovering exact equations, we can introduce the algorithm *Diofantos* for solving the task. The input to the algorithm is the matrix of integers $M = (m_{ij}) \in \mathcal{M}(m, p+1, \mathbb{Z})$, that contains the values of the data table that is the input to the equation discovery task. Note that m denotes the number of observations. At the same time, $p+1$ denotes the number of variables in V: as in the data table, the matrix element m_{ij} corresponds to the i-th observation of the variable x_j, if $j \leq p$, or y, otherwise.

Note that we can rewrite M as a block matrix

$$M = [X|\boldsymbol{y}], \tag{1}$$

where X denotes a matrix from $\mathcal{M}(m, p, \mathbb{Z})$ 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

$$e(X) = \boldsymbol{y}, \tag{2}$$

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 a matrix form as $X \cdot \boldsymbol{c}$, where $\boldsymbol{c}$ is the vector from $\mathbb{Z}^p$. For finding $\boldsymbol{c}$, if it exists, one should solve the system of linear Diophantine equations

$$X \cdot \boldsymbol{c} = \boldsymbol{y}. \tag{3}$$

using well-known and already implemented solvers of systems of linear Diophantine equations [19]. If the system has a solution $\boldsymbol{c} = \begin{bmatrix} c_1 & c_2 & \ldots & c_p \end{bmatrix}^T$, we declare the successful discovery of the linear equation

$$y = \sum_{i=1}^{p} c_i x_i. \tag{4}$$

Otherwise, we report failure.

To allow for discovery of non-linear equations with *Diofantos*, we employ the procedure used in the equation discovery algorithm SINDy [16]. 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 1, implements this procedure in lines 5 to 8: 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 8). 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 a first, new column of X, to allow discovery of equations with linear intercept (line 11). 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 12 to 16).

Algorithm 1 *Diofantos* algorithm for discovery of exact integer equations

Require: 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
Require: The maximal degree $d_{\max}$ of the non-linear terms
Ensure: Equation of the form $y = f(x_1, x_2, \ldots, x_p)$

```
1:  function Diofantos($M$, $d_{\max}$)
2:      $M = [X|\boldsymbol{y}]$
3:      $q = 0$
4:      $T = []$
5:      for $k \in \{1, 2, 3, \ldots, d_{\max}\}$ do
6:          for $(x_{\ell_1}, x_{\ell_2}, \ldots, x_{\ell_k}) \in k$-multicombination$(V \setminus \{y\})$ do
7:              $q = q + 1$
8:              Add a new column $t_q$ to $T$ with element-wise product of the columns corre-
    sponding to $x_{\ell_1}, x_{\ell_2}, \ldots, x_{\ell_k}$
9:          end for
10:     end for
11:     Solve the system of linear Diophantine equations $[\mathbf{1}|T] \cdot \boldsymbol{c} = \boldsymbol{y}$ wrt $\boldsymbol{c}$
12:     if Solution $\boldsymbol{c} = \begin{bmatrix} c_0 & c_1 & \ldots & c_q \end{bmatrix}^T$ exists then
13:         return Equation $y = c_0 + \sum_{i=1}^{q} c_i t_i$
14:     else
15:         return Failed
16:     end if
17: end function
```

For solving the systems of Diophantine equations, we employed the Python package *Diophantine* by [43]. It implements the LLL algorithm to calculate the Hermite normal form of the equations [19].

Let us reconsider the example of the Fibonacci sequence from Table 2 and how *Diofantos* algorithm deals with 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):

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

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

$$c_0 = 0, c_1 = 1, c_2 = 1, c_3 = 0, c_4 = 0, c_5 = 0, \tag{6}$$

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

$$a_n = 0 \cdot 1 + 1 \cdot a_{n-1} + 1 \cdot a_{n-2} + 0 \cdot a_{n-1}^2 + 0 \cdot a_{n-1}a_{n-2} + 0 \cdot a_{n-2}^2 = a_{n-1} + a_{n-2}. \tag{7}$$

Note two important limitations of the *Diofantos* algorithm. First, the *Diofantos* algorithm has the same limitation as SINDy: it can only discover equations, linear with respect to the constant parameters. The second, related to its computational complexity, will be discussed in the continuation.

3.3. Computational Complexity of Diofantos

The procedure for introducing new terms (lines 5–10 in Algorithm 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

$$1 + \sum_{k=1}^{d_{\max}} \binom{p+k-1}{k}, \tag{8}$$

which belongs to $\mathcal{O}(p^{d_{\max}})$. 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 11 in Algorithm 1) is solving the system of linear Diophantine equations using the solver from [19]. The latter is based on the LLL algorithm, which has a time complexity of $O(q^3 m \log(B))$ arithmetic operations [44], 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 the context of *Diofantos* the number of variables in the system corresponds to the number of variables/terms in the discovered equations (q), while the number of equations corresponds to the number of rows in the input matrix M.

Note also 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 $O(q^5 m \log^3(B))$. Considering an input size of $O(qm \log(B))$ and the expectation that $m \ll q$, this results in cubic time complexity relative to the input size. Moreover, the space complexity is in $O(q^2 m \log(B))$ bits, leading to quadratic space complexity relative to the input size. Consequently, the bit-wise computational complexities of *Diofantos* are in $O(p^{5 \cdot d_{\max}} m \log^3(B))$ for time and $O(p^{2 \cdot d_{\max}} m \log(B))$ for space, respectively. Here, we use the same notation as in the previous subsection: p represents the number of input variables, $d_{\max}$ denotes the maximal degree, m signifies the number of observations, and B represents the maximum absolute value of the input data.

When discovering recursive equations of order o, we can assume $p = o + 1$, which leads to the time and space complexities of $O(o^{5 \cdot d_{\max}} N \log^3(B))$ and $O(o^{2 \cdot d_{\max}} N \log(B))$, respectively, where N corresponds to the number of training sequence elements, while B represents the maximum absolute value of the sequence

$$B = \max_{1 \leq i \leq N} |a_i|.$$

For recursive sequences, the input size is $O(N \log(B))$ since the input is the first N sequence elements. So, in this case, and taking into account the expectation that $N \ll o^{d_{\max}}$, the time complexity of *Diofantos* is proportional to the cube of the input size. In contrast, its space complexity is linear regarding the input size. However, we should note that in this case, the crucial factor of complexity is the number of variables q, which is in $O(o^{d_{\max}})$ and, therefore, the complexity relative to the input space might be misleading. What is important is the impact of the recursion order o and the maximal degree of terms $d_{\max}$ on the computational complexity, which was estimated at the beginning of this paragraph.

4. Experimental Evaluation

We empirically evaluate *Diofantos* on 27,370 sequences from the online encyclopedia of integer sequences (OEIS). The first part of this section introduces the experimental setup, the two data sets of integer sequences, the two algorithms for equation discovery compared in the experiments and the algorithm settings, and the measures used to assess the algorithm performance. In the second part, we report and discuss the evaluation results to validate our central conjecture that due to its ability to discover exact equations, *Diofantos* will outperform traditional state-of-the-art algorithms for discovering approximate equations and computer programs for the OEIS sequences.

4.1. Data Sets

We evaluate *Diofantos* on its ability to discover recursive equations in two sets of integer sequences from the online encyclopedia of integer sequences, OEIS [20].

The first data set, referred to as *linrec*, includes 27,236 sequences with known linear recursive equations having constant parameters, indexed by [45]. For each sequence, the index provides a vector of the constant parameters that uniquely identifies the recursive equation for calculating the sequence elements. Consider, for example, coefficients vector $(-1, 0, 2) \in \mathbb{Z}^3$. Its elements identify the recursive equation $a_n = -1 \cdot a_{n-1} + 0 \cdot a_{n-2} + 2 \cdot a_{n-3} = -a_{n-1} + 2a_{n-3}$ of order three. Thus, for each sequence in this data set, we know the ground-truth equation, which allows us to assess the reconstruction abilities of the equation discovery algorithms. The index of linear-recursive sequences on the 4 of March 2023 included 34,371 sequences. However, we have noticed that the corresponding coefficient vectors can be (1) unavailable for 663 sequences, (2) incomprehensible for 4 sequences, (3) difficult to store for 29 sequences and (4) wrong for 6439 sequences. After excluding these three groups, we ended up with 27,236 sequences.

The second, *core* data set that we recreated on the 4 March 2023, included 164 core sequences from the OEIS, indexed by [46]. The core sequences in the OEIS are those that are considered to be fundamental, widely applicable in mathematics and other fields, and often used as building blocks for analyzing and generating other sequences. They are also frequently referenced and have been heavily researched. Examples include trivial sequences, like the sequence of natural numbers, simple sequences with known closed-form and recursive equations, e.g., the Fibonacci and Catalan numbers, and others for which no equations are known or available, such as the sequence of prime numbers. There are 178 core sequences in the OEIS. Our data set does not include seven trivial core sequences that are not labeled as *nice*: these include, e.g., the sequence of natural numbers, two constant sequences of ones and zeroes, and the sequence of negative numbers. Furthermore, our data set does not include seven sequences labeled by the OEIS's keyword *more* denoting sequences in the need for more terms. In sum, the list of sequences included in the *core* data set can be obtained (at the time of conducting our research) with the OEIS search query `keyword:core keyword:nice-keyword:more`. Notably, 30 of these sequences are also already included in the *linrec* data set.

For each sequence, we included its first 200 elements in the corresponding data set. Note that for some of the core sequences less than 200 elements are documented or available: in such cases, we included in the data set all the elements available in the OEIS. The *linrec* data set includes also the coefficient vector for each sequence. While the sequence elements are used as training data for equation discovery, the coefficient vectors are only used to assess the ability of the equation discovery algorithms to reconstruct known equations from data. To support the reproducibility of our results or further experiments with equation discovery, we made the *linrec* and *core* data sets publicly available [47].

Code of our implementation of *Diofantos* and SINDy-based approach, the scripts for our experiments and the corresponding databases of sequences and resulting files are available at our Zenodo [48] repository.

4.2. Compared Algorithms and Their Settings

As already hinted in Section 3.2, we apply *Diofantos* to a given input sequence by restricting the set of variables of the input matrix X to the variables $n, a_{n-1}, a_{n-2}, \cdots, a_{n-o}$ where o is a chosen maximal recursion order. To find as simple equation as possible for a given sequence, we can readily start with the maximal order of 0. We can then increase the complexity of the possible discovered equation by increasing the order of the recursion.

Details for the experimental setup of the *Diofantos* approach given a sequence from the *linrec* data set are presented in Algorithm 2.

Lines 4 and 5 imply increasing the recursion order from 1 to 20. In line 6, we apply *Diofantos* with variables representing recursive terms up to the current maximal order o and we set the *Diofantos* to discover only equations linear in those terms ($d_{max} = 1$). By enforcing the condition $order(e) = o$ in this step, we avoid discovering equations that do not hold for the first few elements of a sequence.

If the condition would not hold, then *Diofantos* would have found such an equation in previous iterations of lower orders which would indicate a flaw in our program. The reason for this happening is that when constructing the system of Diophantine equations of a given order o, any recursive relation of a lower order between the first o terms is ignored. This way, it can happen that for these first terms, the resulting recursive relation of lower order does not hold. The experiment ends when any equation (of the appropriate order) is discovered or when all 20 orders are tried.

For the *core* data set, the experimental setup differs in only a few details and is presented in the form of the pseudocode in Algorithm A1 of Appendix A. The most noticeable difference lies in discovering also equations that are quadratic and cubic multivariate polynomials in the observed variables. This is achieved by changing also the degree d_{max} that is increasing from 1 to 3 (Algorithm A1, line 3).

Another difference includes adding the variable n for discovering, e.g., closed-form equations (Algorithm A1, line 7). Such equations are targeted in the first iteration of the loop (line 6) in the experimental setup which begins with the order 0 instead of 1 as before.

In the *linrec* experiments, we excluded variable n, assuming that the background knowledge is linear-recursive. A slightly more subtle difference is that, here, the order increases only up to 10 (line 5). We decided to reduce the order here since higher degree causes the number of combinations for terms to explode which increases the time complexity of each corresponding iteration. For comparison, for degree 1 and order 20, we have 22 terms (20 recursive terms, one constant 1 and one variable n), while at degree 3 and order 10, we have 364 terms ($\binom{11+3-1}{3} + \binom{11+2-1}{2} + 11 + 1 = 364$).

Algorithm 2 Pseudocode for experiments: *Diofantos* approach for *linrec* data set

Require: Matrix M of observations of the variables $V = \{a_{n-1}, a_{n-2}, \ldots, a_{n-20}, a_n\}$ derived from the first 200 elements of a sequence a_i; the last column of M corresponds to the target variable a_n

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

```
1:  function Diofantos-LINREC(M)
2:      $M = [X|\boldsymbol{y}]$
3:      $X = []$
4:      for $o \in \{1, 2, \ldots, 20\}$ do
5:          Add to $X$ a column of $M$ corresponding to $a_{n-o}$
6:          if Diofantos($[X|\boldsymbol{y}], d_{max} = 1$) returns an equation $e$ and order$(e) = o$ then
7:              return $e : a_n = c_0 + \sum_{i=1}^{o} c_i a_{n-i}$
8:          end if
9:      end for
10:     return Failed
11: end function
```

The pseudocode for the experiments with SINDy is presented in Algorithms A2 and A3 of Appendix A. For the SINDy experiments, we employed its Python implementation `PySINDy` [49,50] and utilized a polynomial library of functions to introduce non-linear terms, mirroring the approach employed by *Diofantos*. Since SINDy performs floating-point arithmetic operations, rather than exact integer arithmetic, we truncated the sequences used for training (lines 2 and 3 in Algorithms A2 and A3, respectively). Each sequence was truncated at the first element whose power to the d_{max} exceeded the precision limit of 10^{16}, which is the precision of floating-point operations. This truncation was necessary due to the transformation of a sequence into the data set, which includes powers to the d_{max} of the elements in the sequence. Incorporating larger elements would be ineffectual, as SINDy would be incapable of accurately measuring the discrepancy between the observed and calculated values of the elements.

To alleviate potential bias in the experimental results, we executed SINDy for each sequence with varying settings of two hyperparameters of the algorithm. The first hyperparameter, the threshold parameter, regulates the strength of regularization in the sparse regression. A higher threshold parameter value results in a sparser regression, leading to the inclusion of fewer terms in the equation. We employed 11 equidistant values for the threshold parameter within the range $[0, 0.9]$ (lines 7 and 12 in Algorithms A2 and A3). The second hyperparameter is a Boolean flag that controls the utilization of ensembles in SINDy. We employed two distinct values for this hyperparameter: no ensembles and ensembles generated through a sampling of matrix rows and columns (lines 8 and 13 in Algorithms A2 and A3). In total, we executed SINDy with 22 distinct hyperparameter settings for each sequence ($11 \times 2 = 22$).

The recursion order limit was set identically to the experiments with *Diofantos*, i.e., to 20 for the *linrec* data set and to 10 for the *core* data set.

Finally, we report two results for SINDy, one for *SINDy-default*, where we only use the default settings of the two hyper-parameters, and one for *SINDy-tuned*, where we try all the settings of the hyper-parameters.

Since the discovered equations from SINDy are approximate, we rigorously validated each equation discovered by SINDy on all the sequence elements included in the data set using exact integer arithmetic. Only equations that passed the validity test (function Validate in lines 10 and 15 in Algorithms A2 and A3) were reported.

For practical reasons, we limited the run-time of each algorithm on a given sequence to two days on an HPC cluster.

4.3. Performance Assessment

We evaluate the performance of the equation discovery algorithm on a data set using two quantitative measures. The first is the number (or ratio) of sequences in the data set for which the algorithm discovered equations. The second is reconstruction rate, i.e., the number (or ratio) of sequences for which the algorithm discovered an equation which is equivalent to the ground-truth equation documented in the OEIS.

The last aspect of performance assessment is qualitative, unsystematic and performed manually. Note that the discovered equations that are not equivalent to the equations documented in the OEIS might be considered to represent novel discoveries. We checked a sample of those and, if considered valid after manual inspection, we decided to report them to the OEIS curators. We report success when the curators accept the identified equation for publication in the OEIS.

4.4. Results on the linrec Data Set

Table 3 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 better also in the number of 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. Comparison of the performance of *Diofantos* and SINDy on the *linrec* data set. The second and third columns report the number of sequences for which the algorithm discovered equations, identical (second column) or different (third column) from the ones reported in the OEIS. The last column reports the number of sequences for which the algorithm failed.

	Success		
Algorithm	**Identical (id)**	**Different (diff)**	**Fail**
Diofantos	9206	15,555	2475
SINDy-default	5678	11,822	9736
SINDy-tuned	6445	12,595	8196

Table 4 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 figure is that more than half of the equations discovered with any of the three algorithms are simpler than the ground-truth ones. While this might be interpreted as a hint to a potential discovery of novel, simpler equations not present in the OEIS encyclopedia, 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 of the sampled equations with the 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.

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 [45]. 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 reminiscent of the ground truth except that it has a twice smaller order. In particular, the ground-truth equation $a_n = 9 \cdot a_{n-4} - 36 \cdot a_{n-8} + 84 \cdot a_{n-12} - 126 \cdot a_{n-16} + 126 \cdot a_{n-20} - 84 \cdot a_{n-24} + 36 \cdot a_{n-28} - 9 \cdot a_{n-32} + a_{n-36}$ has an order of 36, while *Diofantos* discovered the following exact equation of order 18: $a_n = -9 \cdot a_{n-2} - 36 \cdot a_{n-4} - 84 \cdot a_{n-6} - 126 \cdot a_{n-8} - 126 \cdot a_{n-10} - 84 \cdot a_{n-12} - 36 \cdot a_{n-14} - 9 \cdot a_{n-16} - a_{n-18}$.

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 more complex than the ground truth, which is an important and promising result. The other two algorithms based on SINDy lead to such discoveries in relatively rare cases: less than 2% for 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 4. Comparison of the complexity of equations discovered by *Diofantos* and SINDy against the complexity of the ground-truth equation on the *linrec* data set. The second to fourth columns report the comparison concerning the recursion order. The fifth to seventh columns compare the number of nonzero summands on the right-hand side of the equation, with constant terms counted as half. The second and fifth (<) columns report the number of sequences for which the algorithm discovered equations of lower complexity than the ground-truth one. The third and sixth (=) and fourth and seventh (>) columns report the number of sequences for which the algorithm discovered equations of equal complexity (=) or higher complexity (>) than the ground truth, respectively.

Algorithm	**Recursion Order**			**Number of Summands**		
	<	=	>	<	=	>
Diofantos	15,555	9206	0	13,097	9209	2455
SINDy-default	11,688	5690	122	9853	5716	1931
SINDy-tuned	12,226	6493	321	10,353	6579	2108

Table 5 reports on the analysis of the cases when the algorithms failed to discover an exact equation (corresponding to the last column of Table 3). 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 5. The distribution of complexities of the ground-truth equations that were not reconstructed, i.e., the ones for which *Diofantos* and SINDy failed. The second and third columns ($\leq$20 and $>$20) report the numbers of ground-truth equations with orders up to 20 and higher than 20, respectively.

Algorithm	**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 [45]. However, some were also revealed after using both methods hidden in the veil of extra positive results, e.g., finding simpler equations (e.g., for sequence A322829), such as the one reported 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. Note that for the reporting in this section, we included only results for sequences for which no errors were detected from our side.

4.5. Results on the core *Data Set*

Results for *core* data set, as presented in Table 6, 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 truly successful reconstructions, *Diofantos* discovered seven equations for sequences A000058, A000396, A000612, A000798, A001699, A005588, and A055512 that hold for only the selected set of first sequence elements but are very likely to overfit them. Two signs of overfitting are the complexities of the discovered equations and the enormous values of some of the constant coefficients in the equations. The differences between versions *SINDy-default* and *SINDy-tuned* hold only for one sequence, namely A002531, for which *SINDy-default* was unsuccessful.

Table 6. Comparison of the performance of *Diofantos* and SINDy on the *core* data set. Results are grouped by categories. In the case of *Diofantos*, D denotes the number of discovered equations, and SR denotes the number of successfully reconstructed equations, i.e., the number of discovered equations that match known ones from the OEIS.

Seq. Category	**#Sequences**	***Diofantos* [D/SR]**	***SINDy-default***	***SINDy-tuned***
Trivial	4	4/4	4	4
Simple	66	34/32	30	31
Other	94	5/0	0	0
Σ	164	43/36	34	35

Finally, Table 6 shows comparative results in three categories of *core* sequences. We can observe that the differences in the approaches hold only for simple equations. All the overfitted discoveries by *Diofantos* were on core sequences in the last category that includes many sequences for which exact recursive equations has not been found.

4.6. New Discoveries

As mentioned above, we discovered a new equation (or simplification of the previous one) that was previously not documented in the OEIS. Both *Diofantos* and SINDy discovered it. We successfully submitted the simplification to the corresponding OEIS entry at https://oeis.org/A078475 (accessed on 26 November 2024). If any future changes occur to the sequence's entry, our contribution can still be verified in the revision history at revision `#59`, dated 21 September 2023 [51].

Similarly, *Diofantos* discovered the equation $a_n = 2 \cdot n - a_{n-2} + 2 \cdot a_{n-1} - 1$ for sequence entry https://oeis.org/A000330 (accessed on 26 November 2024). However, this discovery is not as significant as the previous one since it can be easily derived from an already-known and documented linear-recursive equation.

4.7. Comparison with Other State-of-the-Art Algorithms

We further compared the performance of *Diofantos* with the performance of the QSynt algorithm [28] on the two data sets of sequences included in our study. Before reporting the results, note that QSynt can discover computer programs for generating the elements of a given sequence. Since computer programs have higher expressive power than recursive equations, we could expect that QSynt should outperform *Diofantos* regarding the number of successful reconstructions.

Table 7 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 the higher constant values can be obtained by combining these values with mathematical operations of, e.g., addition and multiplication, the complexities of such programs render 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 7. Comparison of performance of *Diofantos* and QSynt on *linrec* and *core* benchmarks based on the number of reconstructed equations. The last row (other) corresponds to the number of programs discovered by QSynt for sequences not included in our study. Since 30 core sequences also belong to the data set *linrec* the numbers for QSynt do not sum up to the expected total.

Seq. Category	**QSynt**	***Diofantos***
linrec	14,168	24,761
core	50	36
other	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: $(\text{loop}(\lambda(x,y).x*y,(x+x)+x,1)$ div $\text{loop}(\lambda(x,y).(x*y)+x, x+x,1))$ div $\text{loop}(\lambda(x,y).x*y,x,1)$.

Note that our comparison with QSynt is limited to the sequences in the *linrec* and *core* data sets. The results of QSynt include almost 14 thousand successful reconstructions for other OEIS sequences [28], which are not included in our study.

Finally, we compare the performance of *Diofantos* to the performance of the transformer models [39] for equation discovery on the OEIS sequences (we will refer to their algorithms using the acronym TM-OEIS). Since the personal communication with the authors of TM-OEIS revealed that their model implementation is no longer available, we limited our comparison to the 10 thousand sequences included in their study. We also adopt their performance metrics for comparison, which is predictive accuracy. After discovering a model from the first 15 or 25 sequence elements, they test its predictive accuracy by comparing the model predictions for the next element or the following ten elements (i.e., elements not used in the discovery phase). Since the selection of sequences in the data set is not limited to those with linear-recursive equations, we naturally conducted experiments like with the *core* data set (Algorithm A1).

Table 8 reports the results of the comparison. While *Diofantos* outperforms TM-OEIS when predicting the following ten sequence elements (22% vs. 19% when training on 15 elements and 26% vs. 21% when training on 25), it achieves lower accuracy when predicting the next sequence element (29% vs. 33% when training on 15 elements and 28% vs. 35% when training on 25). This might indicate overfitting by the approximate equations to the training data: the next element can be accurately predicted with an overfitted model, but the latter fails to predict the following sequence elements away from the training ones. Also, when successfully discovered by *Diofantos*, exact equations are very likely robust and equally accurate in predicting an arbitrary number of the following sequence elements. The handicap of *Diofantos* when measuring the overall accuracy is also because failing to discover an exact equation leads to no predictions of the following sequence elements. This is interpreted as zero following elements are correctly predicted, therefore losing any chance of contributing to the overall accuracy of *Diofantos*. In contrast, approximate models might still be able to provide some accurate predictions and, thus, contribute to the overall accuracy of TM-OEIS.

Table 8. Comparison of the performance of *Diofantos* and TM-OEIS on the ten thousand sequences used in [39] according to the predictive accuracy of the discovered equations. The four cases correspond to the combinations of the number of following terms to predict (one or ten), and the number of first elements (15 or 25) of the sequences were used to discover the equations.

Model	$n_{input} = 15$		$n_{input} = 25$	
	$n_{pred} = 1$	$n_{pred} = 10$	$n_{pred} = 1$	$n_{pred} = 10$
TM-OEIS	**33.4**	19.2	**34.5**	21.3
Diofantos	28.9	**22.1**	28.4	**25.6**

4.8. Discussion of Results

As discussed at the end of Section 3, the main limitation of *Diofantos* 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 max recursion order $o_{\max}$, which we set to 20. 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, as discussed in Section 3, it has a significant (cubic) influence on the computational complexity of the crucial step of *Diofantos*, solving the system of Diophantine equations. Second, 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 the discovery of approximate equations.

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 2, where we only iterate through increasing recursive orders used for invoking *Diofantos*, and Algorithm A1, 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.

Empirical analysis of the running times for the two data sets and algorithms, presented in Table 9, confirms the expectations from the theoretical analysis of the computational complexity. The surge in the number of variables for higher degrees in the *core* data set correspondingly escalates the solver's running times required to resolve such extensive systems of Diophantine equations. Consequently, the experimental running times also increase. 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. Finally, the excessive time needed for the successful reconstruction experiment is due to the need to check the exact validity of the discovered equations, which typically involves exact mathematical operations on vast values of the training sequence elements.

Table 9. Comparison of the running times of *Diofantos* and SINDy on *linrec* and *core* data sets in approximate real-time units. A = time needed to finish all experiments, R = time required for all successful reconstruction experiments.

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

4.9. The Utility of Diofantos Outside OEIS

We included a mini set of eight real-world benchmarks to validate the generalizability of *Diofantos* in practical applications outside OEIS and recursive equations for integer sequence. We selected the benchmarks from well-known equalities in basic mathematics, linear algebra, and graph theory. Table 10 presents the eight selected exact equations on four data sets.

Table 10. Blueprint of the eight mathematical benchmarks outside of OEIS. The first column identifies the data set, with variables enlisted in the last column. The third and second column provides the equations and their names, respectively.

Dataset	Name	Equation	Variables
1	Pitagora equation	$a^2 + b^2 = c^2$	a, b, c^2
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.	$\lvert E(W_n)\rvert = 2n$ $\delta(W_n) = 3$ $\Delta(W_n) = n$	$n, \lvert V(W_n)\rvert, \lvert E(W_n)\rvert,$ $\delta(W_n), \Delta(W_n)$

We generated each of the artificial data sets in the following manner. For the first equation, we copied 50 Pitagora 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 final data set, we selected the first 100 wheel graphs W_n, with each graph corresponding to the number of nodes n it contains.

Following the experimental setup used for the *core* OEIS data set, *Diofantos* successfully reconstructed all the equations from Table 10. The result illustrates the ability of *Diofantos* to discover exact mathematical equations.

5. Conclusions

In this study, we present *Diofantos* a novel algorithm for discovering exact equations directly from observed data. It builds upon the well-established SINDy (Sparse Identification of Nonlinear Dynamics) algorithm for equation discovery [16], a recent variant of Lagrange [17]. It diverges from its predecessors by employing a solver of a system of linear Diophantine equations instead of (sparse) linear regression for identifying optimal non-linear combinations of terms.

The experimental evaluation reveals that *Diofantos* significantly outperforms SINDy in discovering exact equations from a selection of linearly recursive sequences within the online encyclopedia of integer sequences (OEIS) [20]. In particular, *Diofantos* successfully reconstructs 90.91% of linear recursive formulas from the first 200 elements of these sequences. This represents a substantial improvement over SINDy's reconstruction rates of 69.91% and 64.25% for *SINDy-default* and *SINDy-tuned*, respectively. Both algorithms displayed comparable reconstruction rates in our analysis of the second selection, referred to as *core*, consisting of more intricate equations. Finally, note that the OEIS curators recognized and validated one discovery made by our proposed algorithm, *Diofantos*, representing a valuable simplification of an already known equation within their extensive database.

There are several potential avenues for further research. Firstly, to improve efficiency, we suggest considering the magnitude of sequence elements when deciding how many to include as input for equation discovery. Different decision heuristics can be applied to trade-off between computational feasibility and the probability of success. Additionally, there is room for improvement of *Diofantos* by investigating deeper into the solver algorithm for Diophantine equations to limit solutions to those with a nonzero term of the highest order. Finally, in the empirical evaluation of *Diofantos* one would replace the manual inspection and validation of results with automated procedures based on symbolic computing for checking the equivalence of the discovered equations with ground truth.

Secondly, *Diofantos* can be applied to discovering or reconstructing relationships among properties of objects in publicly available databases of graphs and knots. Furthermore, this study focused on discovering and reconstructing recursive equations from a single OEIS sequence. A more intriguing task would be identifying relations among different OEIS sequences. To manage a large number of sequences efficiently, one could employ traditional machine learning algorithms for training predictive models and evaluating the relevance of predictive variables to identify strong relationships between sequences in the OEIS, much like the approach presented by [8]. In turn, the recognized relationships might be a foundation for testing conjectures using equation discovery with *Diofantos*. Unlike the approach presented by Davies et al. [8], where conjecture validity is manually checked, these automatically generated tests could potentially lead to new insights and discoveries.

Author Contributions: Conceptualization, B.G. and L.T.; methodology, B.G. and L.T.; software, B.G.; validation, B.G.; formal analysis, B.G.; investigation, B.G.; resources, S.D.; data curation, B.G.; writing—original draft preparation, B.G. and L.T.; writing—review and editing, B.G. and L.T.; visualization, B.G.; supervision, L.T. and S.D.; project administration, L.T. and S.D.; funding acquisition, S.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Slovenian Research Agency via the research program "Knowledge Technologies", P2-0103, and the Gravity project "AI for science", GC-0001.

Data Availability Statement: The original data presented in the study are openly available in Zenodo at https://doi.org/10.5281/zenodo.13767012 (accessed on 26 November 2024).

Acknowledgments: The authors especially appreciate the fruitful discussions within the SHED discussion group (with Jure Brence, Sebastian Mežnar, and Nina Omejc).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Algorithm A1 Pseudocode for experiments: *Diofantos* approach for *core* data set

Require: Matrix M of observations of the variables $V = \{n, a_{n-1}, a_{n-2}, \ldots, a_{n-10}, a_n\}$ derived from 200 first elements of a sequence a_i; the last column of M corresponds to the target variable a_n

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

```
1:  function DIOFANTOS-CORE(M)
2:      $M = [X|\boldsymbol{y}]$
3:      for $d_{\max} \in \{1, 2, 3\}$ do
4:          $X = []$
5:          for $o \in \{0, 1, 2, \ldots, 10\}$ do
6:              if $o = 0$ then
7:                  Add to $X$ the column of $M$ corresponding to $n$
8:              else
9:                  Add to $X$ the column of $M$ corresponding to $a_{n-o}$
10:             end if
11:             if Diofantos($[X|\boldsymbol{y}], d_{max}$) returns an equation $e$ and order($e$) $= o$ then
12:                 return $e$
13:             end if
14:         end for
15:     end for
16:     return Failed
17: end function
```

Algorithm A2 Pseudocode for experiments: *SINDy-tuned* approach for *linrec* data set

Require: Matrix M of observations of the variables $V = \{a_{n-1}, a_{n-2}, \ldots, a_{n-20}, a_n\}$ derived from the first 200 elements of a sequence a_i; the last column of M corresponds to the target variable a_n

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

```
 1: function SINDY-TUNED-LINREC(M)
 2:     $M = \text{Truncate}(M, d_{\max} = 1)$
 3:     $M = [X|\boldsymbol{y}]$
 4:     $X = []$
 5:     for $o \in \{1, 2, \ldots, 20\}$ do
 6:         Add to $X$ a column of $M$ corresponding to $a_{n-o}$
 7:         for $\theta \in \{0, 0.1, 0.2, 0.3, \ldots, 0.9, 1.0\}$ do
 8:             for ens $\in$ {True, False} do
 9:                 $e = \text{SINDy}([X|\boldsymbol{y}], d_{max} = 1, \theta = \theta, \text{ens} = \text{ens})$
10:                 if Validate($e$) then
11:                     return $e : a_n = c_0 + \sum_{i=1}^{o} c_i a_{n-i}$
12:                 end if
13:             end for
14:         end for
15:     end for
16:     return Failed
17: end function
```

Algorithm A3 Pseudocode for experiments: *SINDy-tuned* approach for *core* data set

Require: Matrix M of observations of the variables $V = \{n, a_{n-1}, a_{n-2}, \ldots, a_{n-10}, a_n\}$ derived from 200 first elements of a sequence a_i; the last column of M corresponds to the target variable a_n

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

```
 1: function SINDY-TUNED-CORE(M)
 2:     for $d_{\max} \in \{1, 2, 3\}$ do
 3:         $M = \text{Truncate}(M, d_{\max})$
 4:         $M = [X|\boldsymbol{y}]$
 5:         $X = []$
 6:         for $o \in \{0, 1, 2, \ldots, 10\}$ do
 7:             if $o = 0$ then
 8:                 Add to $X$ the column of $M$ corresponding to $n$
 9:             else
10:                 Add to $X$ the column of $M$ corresponding to $a_{n-o}$
11:             end if
12:             for $\theta \in \{0, 0.1, 0.2, 0.3, \ldots, 0.9, 1.0\}$ do
13:                 for ens $\in$ {True, False} do
14:                     $e = \text{SINDy}([X|\boldsymbol{y}], d_{max} = 1, \theta = \theta, \text{ens} = \text{ens})$
15:                     if Validate($e$) then
16:                         return $e$
17:                     end if
18:                 end for
19:             end for
20:         end for
21:     end for
22:     return Failed
23: end function
```

References

1. Lenat, D.B. The ubiquity of discovery. *Artif. Intell.* **1977**, *9*, 257–285. [CrossRef]
2. Blanchette, J.C.; Kaliszyk, C.; Paulson, L.C.; Urban, J. Hammering towards QED. *J. Formaliz. Reason.* **2016**, *9*, 101–148. [CrossRef]
3. Irving, G.; Szegedy, C.; Alemi, A.A.; Een, N.; Chollet, F.; Urban, J. DeepMath—Deep Sequence Models for Premise Selection. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016 ; Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: New York, NY, USA, 2016; Volume 29, pp. 2243–2253.
4. Paliwal, A.; Loos, S.; Rabe, M.; Bansal, K.; Szegedy, C. Graph Representations for Higher-Order Logic and Theorem Proving. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 2967–2974. [CrossRef]
5. Welleck, S.; Liu, J.; Lu, X.; Hajishirzi, H.; Choi, Y. NaturalProver: Grounded Mathematical Proof Generation with Language Models. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022; Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A., Eds.; 2022; Volume 35, pp. 4913–4927.
6. Bauer, A.; Petković, M.; Todorovski, L. MLFMF: Data sets for machine learning for mathematical formalization. In Proceedings of the 37th International Conference on Neural Information Processing Systems, New Orleans, LA, USA, 10–16 December 2023; NIPS'23; pp. 50730–50741.
7. Trinh, T.H.; Wu, Y.; Le, Q.V.; He, H.; Luong, T. Solving olympiad geometry without human demonstrations. *Nature* **2024**, *625*, 476–482. [CrossRef] [PubMed]
8. Davies, A.; Veličković, P.; Buesing, L.; Blackwell, S.; Zheng, D.; Tomašev, N.; Tanburn, R.; Battaglia, P.; Blundell, C.; Juhász, A.; et al. Advancing mathematics by guiding human intuition with AI. *Nature* **2021**, *600*, 70–74. [CrossRef] [PubMed]
9. Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M.P.; Dupont, E.; Ruiz, F.J.R.; Ellenberg, J.S.; Wang, P.; Fawzi, O.; et al. Mathematical discoveries from program search with large language models. *Nature* **2023**, *625*, 468–475. [CrossRef] [PubMed]
10. Langley, P. Data-driven discovery of physical laws. *Cogn. Sci.* **1981**, *5*, 31–54. [CrossRef]
11. Schmidt, M.; Lipson, H. Distilling Free-Form Natural Laws from Experimental Data. *Science* **2009**, *324*, 81–85. [CrossRef]
12. Todorovski, L.; Džeroski, S. Declarative Bias in Equation Discovery. In Proceedings of the Fourteenth International Conference on Machine Learning, Nashville, TN, USA, 8–12 July 1997; ICML '97; pp. 376–384.
13. Bridewell, W.; Langley, P.; Todorovski, L.; Džeroski, S. Inductive process modeling. *Mach. Learn.* **2008**, *71*, 1–32. [CrossRef]
14. Mundhenk, T.N.; Landajuela, M.; Glatt, R.; Santiago, C.P.; Faissol, D.M.; Petersen, B.K. Symbolic Regression via Neural-Guided Genetic Programming Population Seeding. *arXiv* **2021**, arXiv:2111.00053.
15. Mežnar, S.; Džeroski, S.; Todorovski, L. Efficient generator of mathematical expressions for symbolic regression. *Mach. Learn.* **2023**, *112*, 4563–4596. [CrossRef]
16. Brunton, S.L.; Proctor, J.L.; Kutz, J.N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl. Acad. Sci. USA* **2016**, *113*, 3932–3937. [CrossRef] [PubMed]
17. Džeroski, S.; Todorovski, L. Discovering Dynamics. In *Machine Learning Proceedings 1993*; Morgan Kaufmann: San Francisco, CA, USA, 1993; pp. 97–103.
18. Tanevski, J.; Todorovski, L.; Džeroski, S. Combinatorial search for selecting the structure of models of dynamical systems with equation discovery. *Eng. Appl. Artif. Intell.* **2020**, *89*, 103423. [CrossRef]
19. Havas, G.; Majewski, B.S.; Matthews, K.R. Extended gcd and Hermite normal form algorithms via lattice basis reduction. *Exp. Math.* **1998**, *7*, 125–136. [CrossRef]
20. Sloane, N.J.A. *The On-Line Encyclopedia of Integer Sequences*; The OEIS Foundation Inc.: Monroeville, PA, USA, 2023.
21. McCasland, R.L.; Bundy, A. MATHsAiD: A Mathematical Theorem Discovery Tool. In Proceedings of the 2006 Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, 26–29 September 2006; pp. 17–22.
22. Han, J.M.; Rute, J.; Wu, Y.; Ayers, E.; Polu, S. Proof Artifact Co-Training for Theorem Proving with Language Models. *arXiv* **2021**, arXiv:2102.06203.
23. Caporossi, G.; Hansen, P. Finding relations in polynomial time. In Proceedings of the 16th International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 31 July–6 August 1999; IJCAI'99; Volume 2, pp. 780–785.
24. Wagner, A.Z. Constructions in combinatorics via neural networks. *arXiv* **2021**, arXiv:2104.14516.
25. Colton, S. Computational Discovery in Pure Mathematics. In *Computational Discovery of Scientific Knowledge: Introduction, Techniques, and Applications in Environmental and Life Sciences*; Springer: Berlin/Heidelberg, Germany, 2007; Chapter Computational Discovery in Pure Mathematics; pp. 175–201.
26. Billey, S.C.; Tenner, B.E. Fingerprint Databases for Theorems. *Not. Am. Math. Soc.* **2013**, *60*, 1034. [CrossRef]
27. Luzhnica, E.; Kohlhase, M. Formula Semantification and Automated Relation Finding in the On-Line Encyclopedia for Integer Sequences. In Proceedings of the Mathematical Software–ICMS 2016, Berlin, Germany, 11–14 July 2016; Greuel, G.M., Koch, T., Paule, P., Sommese, A., Eds.; Springer: Cham, Switherland, 2016; pp. 467–475.
28. Gauthier, T.; Urban, J. Learning Program Synthesis for Integer Sequences from Scratch. In Proceedings of the AAAI Conference on Artificial Intelligence, Washington, DC, USA, 7–14 February 2023; Volume 37, pp. 7670–7677. [CrossRef]

29. Koza, J.; Keane, M.; Rice, J. Performance improvement of machine learning via automatic discovery of facilitating functions as applied to a problem of symbolic system identification. In Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA, USA, 28 March–1 April 1993; Volume 1, pp. 191–198.
30. Džeroski, S.; Petrovski, I. Discovering dynamics with genetic programming. In Proceedings of the Machine Learning: ECML-94, Catania, Italy, 6–8 April 1994; Bergadano, F., De Raedt, L., Eds.; Springer: Berlin/Heidelberg, Germany, 1994; pp. 347–350.
31. Märtens, M.; Kuipers, F.; Van Mieghem, P. Symbolic Regression on Network Properties. In Proceedings of the Genetic Programming, Amsterdam, The Netherlands, 19–21 April 2017; McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P., Eds.; Springer: Cham, Switherland, 2017; pp. 131–146.
32. Cranmer, M. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. *arXiv* **2023**, arXiv:2305.01582. [CrossRef]
33. Brence, J.; Todorovski, L.; Džeroski, S. Probabilistic grammars for equation discovery. *Knowl.-Based Syst.* **2021**, *224*, 107077. [CrossRef]
34. Raayoni, G.; Gottlieb, S.; Manor, Y.; Pisha, G.; Harris, Y.; Mendlovic, U.; Haviv, D.; Hadad, Y.; Kaminer, I. Generating conjectures on fundamental constants with the Ramanujan Machine. *Nature* **2021**, *590*, 67–73. [CrossRef] [PubMed]
35. Glasner, K. Data-driven learning of differential equations: Combining data and model uncertainty. *Comput. Appl. Math.* **2023**, *42*. [CrossRef]
36. Martius, G.; Lampert, C.H. Extrapolation and learning equations. *arXiv* **2016**, arXiv:1610.02995.
37. Udrescu, S.M.; Tegmark, M. AI Feynman: A physics-inspired method for symbolic regression. *Sci. Adv.* **2020**, *6*, eaay2631. [CrossRef]
38. Kusner, M.J.; Paige, B.; Hernández-Lobato, J.M. Grammar Variational Autoencoder. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; Precup, D., Teh, Y.W., Eds.; Proceedings of Machine Learning Research, Volume 70, pp. 1945–1954.
39. D'Ascoli, S.; Kamienny, P.A.; Lample, G.; Charton, F. Deep symbolic regression for recurrence prediction. In Proceedings of the 39th International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022; Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S., Eds.; Proceedings of Machine Learning Research—PMLR, 2022; Volume 162, pp. 4520–4536.
40. Kamienny, P.A.; d'Ascoli, S.; Lample, G.; Charton, F. End-to-end Symbolic Regression with Transformers. In Proceedings of the Advances in Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022; Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A., Eds.; Curran Associates, Inc.: New York, NY, USA, 2022; Volume 35, pp. 10269–10281.
41. Valipour, M.; You, B.; Panju, M.; Ghodsi, A. SymbolicGPT: A Generative Transformer Model for Symbolic Regression. *arXiv* **2022**, arXiv:2106.14131. [CrossRef]
42. Wolfram R., Inc. *Mathematica*, Version 14.1; Wolfram Research, Inc.: Champaign, IL, USA, 2024.
43. Close, T.G. Diophantine: A Python Package for Finding Small Solutions of Systems of Diophantine (Integer Algebra) Equations [Software]. 2017. Available online: https://pypi.org/project/Diophantine/ (accessed on 26 November 2024).
44. Galbraith, S.D. Lattice basis reduction. In *Mathematics of Public Key Cryptography*; Cambridge University Press: Cambridge, UK, 2012; pp. 347–365.
45. Sloane, N.J.A.; The OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences Wiki, Index to OEIS: Section Rec. 2023. Available online: https://oeis.org/wiki/Index_to_OEIS:_Section_Rec (accessed on 4 March 2023).
46. Sloane, N.J.A.; The OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences Wiki, Index to OEIS: Section Cor. 2023. Available online: https://oeis.org/wiki/Index_to_OEIS:_Section_Cor (accessed on 4 March 2023).
47. Gec, B. The Databases of Linear-Recursive and "Core" Sequences [Data]. 2024. Available online: https://zenodo.org/records/13767012 (accessed on 26 November 2024).
48. Gec, B. B0Gec/Diofantos: Diophantos First Paper Release for Zenodo. 2024. Available online: https://zenodo.org/records/13692310 (accessed on 26 November 2024).
49. de Silva, B.; Champion, K.; Quade, M.; Loiseau, J.C.; Kutz, J.; Brunton, S. PySINDy: A Python package for the sparse identification of nonlinear dynamical systems from data [software]. *J. Open Source Softw.* **2020**, *5*, 2104. [CrossRef]
50. Kaptanoglu, A.A.; de Silva, B.M.; Fasel, U.; Kaheman, K.; Goldschmidt, A.J.; Callaham, J.; Delahunt, C.B.; Nicolaou, Z.G.; Champion, K.; Loiseau, J.C.; et al. PySINDy: A comprehensive Python package for robust sparse system identification [software]. *J. Open Source Softw.* **2022**, *7*, 3994. [CrossRef]
51. Vlieger, M.D.; Spezia, S.; Marcus, M.; Howroyd, A.; Gec, B.; Kotesovec, V.; Schoenfield, J.E.; Pol, O.E.; Cox, R.; Sloane, N.J.A. Revision History for A078475. 2023. Available online: https://oeis.org/history/view?seq=A078475&v=59 (accessed on 26 November 2024).