



Probabilistic grammars for modeling dynamical systems from coarse, noisy, and partial data

Nina Omejc^{1,2} · Boštjan Gec^{1,2} · Jure Brence^{1,2} · Ljupčo Todorovski^{1,3} ·
Sašo Džeroski¹

Received: 13 March 2023 / Revised: 30 October 2023 / Accepted: 14 February 2024 /
Published online: 29 May 2024
© The Author(s) 2024

Abstract

Ordinary differential equations (ODEs) are a widely used formalism for the mathematical modeling of dynamical systems, a task omnipresent in scientific domains. The paper introduces a novel method for inferring ODEs from data, which extends ProGED, a method for equation discovery that allows users to formalize domain-specific knowledge as probabilistic context-free grammars and use it for constraining the space of candidate equations. The extended method can discover ODEs from partial observations of dynamical systems, where only a subset of state variables can be observed. To evaluate the performance of the newly proposed method, we perform a systematic empirical comparison with alternative state-of-the-art methods for equation discovery and system identification from complete and partial observations. The comparison uses Dynobench, a set of ten dynamical systems that extends the standard Strogatz benchmark. We compare the ability of the considered methods to reconstruct the known ODEs from synthetic data simulated at different temporal resolutions. We also consider data with different levels of noise, i.e., signal-to-noise ratios. The improved ProGED compares favourably to state-of-the-art methods for inferring ODEs from data regarding reconstruction abilities and robustness to data coarseness, noise, and completeness.

Keywords System identification · Equation discovery · Symbolic regression · Partial observability · Probabilistic context-free grammars · Ordinary differential equations

Editors: Dino Ienco, Roberto Interdonato, Pascal Poncelet.

✉ Nina Omejc
nina.omejc@ijs.si

¹ Department of Knowledge Technologies, Jožef Stefan Institute, 1000 Ljubljana, Slovenia

² Jožef Stefan International Postgraduate School, 1000 Ljubljana, Slovenia

³ Faculty of Mathematics and Physics, University of Ljubljana, 1000 Ljubljana, Slovenia

1 Introduction

Dynamical systems change their state, i.e., evolve through time. Their temporal evolution is commonly modeled by ordinary differential equations (ODEs) of the form $\dot{\mathbf{u}} = f(\mathbf{u}, \boldsymbol{\theta})$, where $\mathbf{u}$ is the d -dimensional vector of state variables, $\dot{\mathbf{u}}$ is the vector of their time derivatives describing the change of their values through time, $f : \mathbb{R}^d \times \mathbb{R}^p \rightarrow \mathbb{R}^d$ is a function, referred to as *model structure*, and $\boldsymbol{\theta} \in \mathbb{R}^p$ are constant *model parameters*. The solution of such ODEs is obtained from an initial state $\mathbf{u}_0 = \mathbf{u}(t_0)$, i.e., the values of the state variables at an initial time point t_0 , by numerical simulation (Hindmarsh et al., 2005) and has a form of a state trajectory $\mathbf{u}(t)$, specifying the values of the state variables at all time points $t > t_0$.

The mathematical modeling of an observed dynamical system is an inverse problem. Given the trajectory $\mathbf{u}(t)$, we aim at finding *appropriate* model structure f and model parameters $\boldsymbol{\theta}$. The simulated trajectory produced by the model should closely match the observed ones and simpler model structures are preferred over more complex ones. Classical approaches to this task, also called system identification, assume that the model structure f is provided at input. Often it is also assumed to be linear (i.e., the right-hand sides of the ODEs are linear combinations of state variables). Alternatively, it is derived from the first principles in the domain of study, where the dynamical systems are studied.

In contrast with classical approaches, equation discovery (or symbolic regression) approaches aim to infer *both* f and $\boldsymbol{\theta}$ from data. They primarily aim to discover algebraic equations, but some are also used to learn ODEs. The most straightforward approach to learn ODEs is to numerically calculate the time derivatives of the state variables $\dot{\mathbf{u}}$ and add them to the set of observed variables. Consequently, ODEs can be discovered as algebraic equations with dependent variables on the right-hand side corresponding to the time derivatives. For example, Lagrange (Džeroski & Todorovski, 1993) and SINDy (Brunton et al., 2016; de Silva et al., 2020) follow this approach and use linear regression variants to learn ODEs that are linear in the model parameters $\boldsymbol{\theta}$.

This simple approach has two limitations. First, calculating derivatives is numerically unstable (Ramm and Smirnova 2001), leading to high sensitivity to noisy measurements. Second, all the state variables have to be observed. To address these limitations, Pret (Stolle and Bradley 2007), ProBMot (Čerepnalkoski 2013), and GPoM (Mangiarotti et al., 2012) employ numerical simulation of ODEs. First, the simulation increases the accuracy of parameter estimation and robustness to noise. Additionally, ProBMot uses numerical simulation to solve the problem of partial observability, as the missing measurements can be estimated via simulation from an initial value, which is treated as another constant parameter to be fitted with numerical optimization. Other methods, such as L-ODEfind (Somacal et al., 2022) and GPoM (Mangiarotti et al., 2012), can learn polynomial ODEs from partial observations by transforming the first-order ODEs to higher-order differential equations.

Note that the adaptations of the symbolic regression methods towards learning ODEs affect only how constant parameters $\boldsymbol{\theta}$ are fitted against data. They use the same procedures for searching through the space of candidate algebraic equation structures f for learning ODEs. To improve the efficiency of the search, equation discovery methods make general assumptions about the space of candidate equations or allow the user to specify domain-specific constraints on this space. SINDy and Lagrange assume that the candidate structures are linear combinations of parameter-free nonlinear terms allowing users to specify a set of functions for building non-linear candidate terms and limiting their degree. In DSO (Mundhenk et al., 2021), the user can select the operators and functions for building equations. ProBMot (Čerepnalkoski 2013) allows the user to specify entities and dynamic

processes that model the interactions between entities in dynamic systems in the domain of interest. ProGED (Brence et al., 2021), a recently proposed method for equation discovery, allows users to specify domain-specific knowledge as a probabilistic grammar defining the space of candidate equation structures.

ProGED (Brence et al., 2021) has not yet been used to learn ODEs from data. In this paper, we propose two extensions of ProGED for discovering ODEs. The first extension introduces new variables for equation discovery by calculating the numerical derivatives of the state variables. The second extension employs methods for numerically simulating ODEs that can be applied to partially observed systems.

We perform a systematic comparative analysis of the performance of the newly proposed extensions of ProGED with alternative state-of-the-art methods. All experiments aim to reconstruct known ODEs of dynamical systems from synthetic data obtained by simulating the known ODEs. Our study uses Dynobench, consisting of ten dynamical systems, seven from the Strogatz benchmark (Strogatz 2018) and three other systems often used to evaluate system identification methods. We assess the impact of three aspects of data imperfection on the performance of the evaluated methods. The first is the coarseness, i.e., the temporal resolution and the length of the simulation. The second is the noise level in terms of the signal-to-noise ratio (SNR). The third is observability, i.e., whether all the system variables are measured. When performing reconstruction experiments from complete observations, we compare ProGED (Brence et al., 2021) to DSO (Mundhenk et al., 2021) and SINDy (Brunton et al., 2016), while in the partial observability experiments, we compare it to L-ODEfind (Somacal et al., 2022) and GPoM (Mangiarotti et al., 2012).

In Sect. 2, we review the related work on inferring ODEs from data. We introduce the extensions of ProGED for ODE discovery and the background knowledge used for system identification in Sect. 3. In Sect. 4, we present the benchmark and the setup of the computational experiments that compare the performance of the different methods, while Sect. 5 reports upon the results of the comparative analysis of the methods. Finally, we summarize our contributions and outline the directions for further research in Sect. 6.

2 Related work

Several methods, based on fast and simple linear regression, have been developed for identifying ODEs that are linear in the parameters. They numerically calculate the time derivatives of the state variables. Lagrange (Džeroski & Todorovski, 1993) is one of the earliest methods, which introduces new higher-order terms with the multiplication of the state variables and uses linear regression to find the constant parameters of the polynomial ODEs involving these terms.

SINDy (Sparse Identification of Nonlinear Dynamics) (Brunton et al., 2016) extends the Lagrange approach by introducing new terms with custom basis functions. Sparse regression techniques are additionally used to help control the complexity of discovered equations and reduce over-fitting.

L-ODEfind (Latent ODE finder) (Somacal et al., 2022) is based on sparse linear regression on terms generated by polynomial basis functions and relies on numerical differentiation. It tackles the partial observability problem by rewriting a system of n first-order ODEs to a single n -th order ODE, the $n - 1$ unobserved variables being implicitly included and their values numerically approximated as higher-order derivatives. Since the inverse

transformation to the first-order ODEs is not uniquely determined, L-ODEfind is useful for predictive modeling, but ultimately inappropriate as a system identification framework.

Generalized Polynomial Modeling, GPoM (Mangiarotti et al., 2012; Mangiarotti and Huc, 2019), is a method for polynomial structure selection of dynamical systems. The method first generates an equation for each state variable by including all possible monomials from a user-defined set of system variables and a maximum polynomial degree. Then, iteratively, parameters are estimated and the leave-one-out approach is used to estimate the relative contribution of each monomial. The monomial with the smallest contribution is removed, and the process is repeated until all monomials have been removed or until a desired number of monomials remain. The possible models for each equation are then combined in a combinatorial manner, and a numerical integration procedure (ODE simulation) is used to obtain a more precise estimate of the parameters. In a partial observability scenario, GPoM also uses higher-order derivatives as a proxy for missing variables, however, GPoM does not output a model with a single, higher-order ODE like L-ODEfind, but rather uses differential embedding to construct a system of first-order ODEs.

Deep Symbolic Optimization, DSO (Petersen et al., 2019; Mundhenk et al., 2021), employs reinforcement learning to train a recursive neural network (RNN) capable of generating symbolic expressions. DSO uses a unique risk-seeking policy gradient that rewards best-case performance instead of expected performance. It additionally utilizes multiple rounds of genetic programming between RNN iterations to enhance performance. Moreover, it can incorporate restricted forms of background knowledge as its priors and constraints. To discover ODEs, DSO must currently rely on numeric differentiation.

Lastly, the transformer architecture has also been used on symbolic regression tasks (Kamienny 2022). The method uses a transformer-based language model to directly predict the full mathematical expression, constants included. The model is trained using a supervised learning approach, where the ground truth mathematical expressions are used to train the model. It is designed for discovering algebraic equations and can handle dynamical system identification via numerical differentiation.

3 Methodology

ProGED discovers equations by following the generate-and-test paradigm. In the generate phase, ProGED addresses the task of structure identification, in which candidate equations (structures) are constructed. The test phase performs parameter estimation, in which the values of unknown numeric parameters in the equations are fit to data. Among a large number of tested equations, ProGED chooses the ones with the lowest error-of-fit. ProGED composes candidate equations from algebraic expressions, sampled from a probabilistic context-free grammar (PCFG).

A context-free grammar (CFG) is defined by the tuple $(\mathcal{T}, \mathcal{N}, \mathcal{R}, S)$. When defining arithmetic expressions, the set of terminal symbols $\mathcal{T}$ consists of symbols representing variables (e.g., x , y), operators or functions (e.g., $+$, $\cdot$, $\sin$), and constant parameters (c). The nonterminal symbols in $\mathcal{N}$ do not appear in expressions, but represent higher-level concepts in the language of mathematics, such as polynomials, monomials or terms. The set $\mathcal{R}$ contains production (rewrite) rules $A \rightarrow \alpha_1 \dots \alpha_k$, where $A \in \mathcal{N}$ and $\alpha_i \in \mathcal{N} \cup \mathcal{T}$. A production rule specifies how to replace a particular nonterminal symbol with a string of nonterminal and terminal symbols. In a probabilistic context-free grammar, each production rule is assigned a probability, so that the probabilities of all production rules

with the same nonterminal symbol on the left-hand side (LHS) sum up to 1. Examples of grammars that were used in our experiments are shown in Table 1 and the Tables in Appendix A.

The generation of a random expression with a PCFG begins from a string (starting symbol) S and proceeds by successively applying production rules to the string until only terminal symbols remain. Whenever more than one rule applies, we randomly choose a rule according to the probabilities. The final result of one instance of the sampling process is an arithmetic expression, which we transform to its canonical form by using the symbolic mathematics engine SymPy (Meurer et al., 2017).

Besides acting as a generator of expressions, a grammar is a powerful way of encoding background knowledge. Note that a PCFG defines a probability distribution over the space of candidate expressions, which allows the user to impose an inductive bias by manipulating the production probabilities. For example, we can manipulate the complexity of generated equations through the probabilities of recursive productions, or express a bias towards trigonometric functions by raising their respective probabilities. In the absence of background knowledge, we can use a universal grammar for generating an arbitrary expression, composed of the four basic operations (+, −, *, /), as well as arbitrary functions.

After generation, a candidate equation contains generic constants (denoted by c), the values of which must be fitted to data. Since the equations are, in general, non-linear in their parameters, a universal optimization algorithm is used to minimize the error-of-fit to the data, which can be computationally demanding, but is more flexible than approaches based on linear regression. ProGED uses differential evolution (Storn and Price 1997) algorithm for numerical optimization to fit the values of the constants in an expression to the provided data.

Table 1 An example of a generic grammar for mathematical expressions appearing in models of dynamical systems, used to generate the right-hand sides of ODEs

$\mathcal{N} = \{E, D, T, R, M, V\}$		
$\mathcal{T} = \{+, *, /, \sin, \cos, (,), c, x, y\}$		
$\mathcal{R} =$		
$E \rightarrow E + T$ [0.6]	$ T/(D)$ [0.15]	$ T$ [0.25]
$D \rightarrow D + T$ [0.5]	$ T$ [0.5]	
$T \rightarrow T * V$ [0.3]	$ T * R$ [0.1]	$ c$ [0.6]
$R \rightarrow \sin(M)$ [0.5]	$ \cos(M)$ [0.5]	
$M \rightarrow M * V$ [0.5]	$ c$ [0.5]	
$V \rightarrow x$ [0.5]	$ y$ [0.5]	
$S = E$		

Production rules with the same nonterminal on the left-hand side are separated by a vertical line (|). The probability of each production rule is given in square brackets. The grammar can generate expressions (E) involving linear combinations of multiplicative terms (T), composed by multiplying variables (V) or trigonometric terms (R). Trigonometric terms include sines and cosines of monomials (M) of the state variables (V). Moreover, the grammar can generate expressions involving the division of linear combinations of multiplicative terms (T) with terms in the denominator (D), where D can be a linear combination of terms T . The example grammar refers to a system with two state variables, x and y , but can be easily extended to an arbitrary number of state variables by adding new production rules to the nonterminal V .

3.1 Numeric differentiation and algebraic equation discovery

The task of discovering ODEs can be transformed into a task of discovering algebraic equations by numerically calculating the derivatives of the state variables. These time derivatives are then considered as dependent variables and placed in the LHS of algebraic equations to be discovered. In that case, we estimate the parameter values by minimizing the difference L between the observed (calculated) and predicted values of the time derivatives of the state variables,

$$L = \sqrt{\frac{1}{n} \sum_{i=1}^n (\dot{u}(t_i) - \hat{u}(t_i))^2} , \quad (1)$$

where $\dot{u}(t_i)$ represents the value of the time derivative of variable u at time t_i , numerically calculated from observed data, $\hat{u}$ represents the corresponding predicted value, obtained by evaluating the candidate equation, and n is the number of observed time points.

This simple transformation is commonly used by ODE discovery approaches. However, its use is problematic if we deal with coarsely sampled measured values of the system variables and high levels of noise. Also, this approach is only possible when the measurements of all state variables are readily available, i.e. the system at hand is fully observable.

3.2 Discovery of differential equations with direct simulation

To address the limitations of numerical differentiation, we introduce an approach to ODE discovery based on simulating differential equations. During each step of parameter estimation, we must compute the error of the candidate equation with a given set of parameter values. To obtain this, we solve the initial-value problem by performing a full simulation of the system of ODEs, using the LSODA algorithm implemented in the function `odeint` from the SciPy library (Hindmarsh 1983; Virtanen et al., 2020). We define the error as the root-mean-squared error of the simulated trajectory, with respect to the true trajectory of the observed variables. In other words, we minimize the error

$$L = \sqrt{\frac{1}{|\mathcal{U}_{\text{obs}}| \cdot n} \sum_{u \in \mathcal{U}_{\text{obs}}} \sum_{i=1}^n (u(t_i) - \hat{u}(t_i))^2} , \quad (2)$$

where $\mathcal{U}_{\text{obs}}$ is the set of all *observed* variables, $u(t_i)$ represents the observed value u at time t_i , $\hat{u}(t_i)$ represents the corresponding simulated value (i.e., the value obtained by simulating the candidate equation), and n is the number of observed time points.

3.3 Parallel computation

Most existing approaches to equation discovery are difficult to parallelize. In contrast, ProGED uses a Monte Carlo algorithm to sample expressions from a PCFG. Thus, each run of the procedure is completely independent of the results of any previous runs. This allows for parallelization of parameter estimation, by far the most computationally

demanding step in ProGED, especially when identifying a partially-observed system of ODEs. Accelerating this step is critical for managing the computation time of ProGED. The easy parallelization and the ability to make good use of available high-performance-computing resources is therefore an important advantage of our approach.

3.4 Background knowledge

ProGED uses Monte-Carlo sampling to search the space of possible systems of ODEs. Since this search is not guided, the method works best when the search space is as constrained as possible. Generally, the search space is constrained based on various types of background knowledge, from general modeling principles, such as the parsimony principle, to domain-specific or even problem-specific knowledge. ProGED employs PCFGs as a robust and powerful framework for expressing different types of background knowledge and imposing both hard constraints (through the PCFG structure) and soft constraints (through production probabilities).

In this study, we aim to discover models of dynamical systems, which already provides some background knowledge in itself. To demonstrate how background knowledge can be expressed with PCFGs, we detail some of the background knowledge for modeling dynamical systems and design a grammar that expresses the general knowledge as follows:

1. A dynamical system is described by a system of 1st-order ODEs.
2. The right-hand side of each ODE is a linear combination of terms.
3. Since each term has its own multiplicative numerical constant, there is no need to include the subtraction operation.
4. Terms are most commonly low-order monomials of state variables.
5. Rarely, trigonometric functions appear, in particular sine and cosine.
6. The arguments of trigonometric functions tend to be linear functions of state variables and/or time.
7. Rarely, terms can be rational functions.
8. The terms in the numerator and denominator of rational functions tend to be low-order polynomials.
9. Numerical constants can appear in trigonometric and rational functions, making the expressions nonlinear in parameters.

We present a PCFG that expresses this background knowledge for modeling dynamical systems in Table 1. The grammar has three production rules for the starting symbol E (*expression*), which generate the sum of a number of terms, which are either ordinary terms derived by T , or rational functions in the form of T/D . The two productions for D (*denominator*) derive the denominator of a rational function as a sum of terms. The productions for T (*term*) generate a monomial, composed of variables and/or trigonometric functions. The productions for R and M derive the trigonometric factors as the sine or cosine of a product of state variables. Finally, the production rules for V generate state variables. The presented grammar generates expressions, following the domain knowledge above. A system of ODEs is composed by independently generating an expression for the right-hand side of each ODE in the system.

The choice of probabilities in the PCFG warrants discussion as well, since it allows us to express soft constraints on the space of equations and decide the level of parsimony

of generated expressions. In the grammar in Fig. 1, we set the probability of recursion in the rules for E relatively high (0.6), in order to generate expressions with several terms. On the other hand, the probability of recursion in the production rules for T is lower ($0.3 + 0.1 = 0.4$), as we prefer lower-order terms. We set the probabilities of rational functions (0.15) and trigonometric functions (0.1) low to reflect their relative rarity in models of general dynamical systems. We set the probabilities in the presented grammar based on past experience with modeling dynamical systems, as well as by generating samples of random expressions and observing their properties.

The presented grammar is designed to constrain the space of expressions as much as possible, while still describing most dynamical systems. Nevertheless, some dynamical systems may be described by systems of ODEs that fall outside the space described by this grammar, which would preclude their discovery with ProGED. This risk is common when using hard constraints in equation discovery. On the other hand, the presented grammar still generates many types of expressions, including the entire class of rational functions. A more limited grammar can be designed to describe more specific types of dynamical systems by altering the structure of the grammar and the values of its probabilities.

4 Experimental evaluation

In this section, we present the details of the experimental evaluation we have conducted. We begin by presenting the data used in the experiments. We then describe the background knowledge used together with the data. This is followed by a description of the experimental evaluation, performed separately under full and partial observability. We conclude this section with a discussion of the performance measures we have used to compare the different ODE discovery methods.

4.1 Data

The benchmark we used for evaluation contains trajectories of ten dynamical systems, which describe the temporal evolution of the state variables of each system. Models of the ten dynamical systems are shown in Table 2. Seven out of the ten systems come from the Strogatz benchmark (Romano et al., 2021; Strogatz 2018), currently widely used for system identification (see e.g. (La Cava et al., 2021)). Additionally, we have added three models of dynamical systems: a model of coupled phase oscillators (cphase); the Stuart-Landau (stl) oscillator (a normal form of a Hopf bifurcation); and the Lorenz chaotic oscillator (lorenz), which have often been used in system identification experiments (Kuznetsov et al., 1998; Stankovski et al., 2014; Strogatz 2018; Brunton et al., 2016) and add some variety to the benchmark set of dynamical systems. The system cphase, for example, introduces time dependency on the right-hand side, a property of non-autonomous systems, while lorenz contains three state variables. We generated the data by directly simulating the models using the LSODA algorithm (with parameters set to $|rtol|=10e-12$, $|atol|=10e-12$). The reason we did not use the Strogatz benchmark data as provided is that while testing the benchmark, we noticed that the Bogacki-Shampine method used for simulation did not provide adequate accuracy for certain dynamical systems. For example, the simulation of the Van der Pol oscillator, using the fourth set of initial values, resulted in unwanted oscillations that were not present when more precise numerical ODE solvers were used. Moreover, the benchmark contains trajectories with only 100 time points, sampled at a 0.1 Hz

Table 2 Description of ten dynamical systems, including their model (system of ODEs), the parameters we have chosen for each system, and the number of terms present in each ODE

System name	System model	Chosen parameters	# Terms
bacterial respiration (<i>bacres</i>)	$\dot{x} = B - x - \frac{xy}{q^2+1}, \dot{y} = A - \frac{xy}{q^2+1}$	$A = 10, B = 20, q = 0.5$	(3, 2)
bar magnets (<i>barmag</i>)	$\dot{x} = K \sin(x - y) - \sin(x), \dot{y} = K \sin(y - x) - \sin(y)$	$K = 0.5$	(2, 2)
coupled phase oscillator (<i>cphase</i>)	$\dot{x} = W(t) + A \sin(x) + B \sin(y), \dot{y} = C \sin(x) + D \sin(y) + E$	$W(t) = 2 - 0.5 \sin(2\pi \cdot 0.0015t), A = B = 0.8, C = 0, D = 0.6, E = 4.53$	(4, 2)
glider	$\dot{x} = -\sin(y) - Dx^2, \dot{y} = -\frac{\cos(y)}{x} + x$	$D = 0.05$	(2, 2)
Lorenz oscillator (<i>lorenz</i>)	$\dot{x} = \sigma(y - x), \dot{y} = x(\rho - z) - y, \dot{z} = xy - \beta z$	$\sigma = 10, \rho = 28, \beta = \frac{8}{3}$	(2, 3, 2)
Lotka-Volterra (<i>lv</i>)	$\dot{x} = x(A - x - By), \dot{y} = y(C - x - y)$	$A = 3, B = 2, C = 2$	(3, 3)
Van der Pol (<i>vdvp</i>)	$\dot{x} = y, \dot{y} = -x - \mu(x^2 - 1)y$	$\mu = 2$	(1, 3)
predator-prey (<i>predprey</i>)	$\dot{x} = x(b - x - \frac{y}{1+x}), \dot{y} = y(\frac{x}{1+x} - ay)$	$b = 4, a = 0.075$	(3, 2)
shear flow (<i>shearflow</i>)	$\dot{x} = \cot(y) \cos(x), \dot{y} = (\cos^2(y) + A \sin^2(y)) \sin(x)$	$A = 0.1$	(1, 2)
Stuart-Landau (<i>stl</i>)	$\dot{x} = ax - \omega y - x(x^2 + y^2), \dot{y} = \omega x + ay - y(x^2 + y^2)$	$a = 1, \omega = 3$	(4, 4)

rate. The coarse sampling of data can significantly affect the performance of system identification methods. For example, some oscillators do not yet complete their limit cycles during this period, so their main characteristic stays hidden. For these reasons, we have decided to improve upon the Strogatz benchmark and make our extended benchmark available on the digital repository Zenodo under the name Dynobench [10.5281/zenodo.10041312](https://zenodo.org/record/10041312).

Analogously to the Strogatz benchmark, Dynobench includes coarse data of only 100 time points sampled at 0.1 Hz (hereafter called "small data"). Additionally, we created densely sampled data with 2000 time points, sampled at 0.01 Hz (hereafter called "large data"). Each system of ODEs was simulated four times, each time with a different set of initial values of the state variables, using the parameter values reported in Table 2. Python code for the simulation of dynamical systems is included in the Dynobench repository.

We also evaluated the methods on clean (noise-free) data, noisy data with a 30 dB signal-to-noise ratio (SNR), where the signal was a thousand times stronger than the noise, and data with a 13 dB SNR (signal twenty times stronger than noise). We added Gaussian noise $U_{noise}[V] \sim \mathcal{N}(0, \sqrt{P_{noise}[W]})$, with mean $\mu = 0$ and standard deviation $\sigma^2 = \sqrt{P_{noise}[W]}$. The $P_{noise}[dB]$ was calculated based on the power of the signal and the desired SNR, following $P_{noise}[dB] = P_{signal}[dB] - SNR[dB]$.

4.2 Background knowledge

The use of background knowledge is an important aspect of equation discovery, but can be difficult to take into account in comparative benchmark experiments. To facilitate its use, we equip the problems in Dynobench with explicit background knowledge. We classify each of the ten dynamical systems into one of three classes that describe its type and define its background knowledge:

1. *State oscillators and population models* include systems *lv*, *predprey*, *vdp*, *stl*, *lorenz* (see Table 2). The right-hand sides of the ODEs for these systems are polynomials, most commonly composed of two or three terms. The terms are monomials of the state variables, typically of order 1–3. An exception that often appears in population models is the Monod term (fraction of the form $\frac{V}{V+c}$ (Monod, 1949)). Interaction terms (terms involving two or more state variables) often appear in the same form in more than one ODE in the systems of ODEs.
2. *Phase oscillators* include the systems *cphase* and *barmag*. Phase oscillators are oscillatory systems, which we observe through the phases of their state variables, instead of the state variables themselves. Consequently, the right-hand sides of the ODEs are linear combinations of trigonometric functions. Sine and cosine are the most common, but other trigonometric functions may appear as well. The functions may be phase-shifted, which is realized by an additive constant in the function's argument. The arguments of the trigonometric functions tend to be linear functions of the phase variables.
3. *General dynamical systems* include *bacres*, *glider* and *shearflow*. Systems that do not fall into any of the previous two classes tend to follow the general background knowledge for modeling dynamical systems, outlined in Sect. 3.4.

4.3 Full observability

To assess the system identification performance of ProGED on fully observable systems, we compared it to the DSO and SINDy methods on the Dynobench benchmark. When evaluating each method, we have done two experiments. First, we tried to use as much of the available background knowledge as possible and second, we left the model search space unconstrained. The first experiment was named *constrained* and second *unconstrained*. The configuration of each method is described in the following paragraphs, starting with the description of the constrained experiments.

For ProGED, we constructed a grammar for each of the three dynamical system categories in Dynobench, based on the background knowledge outlined in Sects. 3.4 and 4.2. We adapted the grammars for the specific benchmark problems by modifying the productions that generate variables (typically productions for V), so as to generate the correct number of variables with the right names. We always used a uniform distribution over the production rules $V \rightarrow v$ that generate the variables. Other than the production rules for V , the structure of the grammars and their production probabilities were fixed for all the benchmark problems in the same category.

First, to identify the *lv*, *predprey*, *vdp*, *stl* and *lorenz* systems, we created a grammar for state oscillators (see Table 7). The grammar generates polynomials with a relatively high probability of generating more terms (0.6, same as in the grammar in Table 1). Each term is the product of a number of state variables and may involve up to one Monod transformation of a state variable. The Monod function is relatively rare (with a probability of 0.1). The probability of recursion in multiplication is low (0.35), to ensure the rarity of higher-order terms.

The second grammar, presented in Table 8) in the Appendix, expresses the background knowledge about phase oscillators. It generates linear combinations of sines and cosines, which use linear combinations of variables as their arguments. We used this grammar for the *cphase* and *barmag* systems.

The third grammar we use is the generic grammar presented in Table 1 and described in Sect. 3.4. It generates a large repertoire of terms involving addition, multiplication, division, and trigonometric functions. It involves only general knowledge about dynamical systems and is used when more specific background knowledge is not available, i.e., for the *bacres*, *glider* and *shearflow* systems.

Using the three grammars, we generated 2500 candidate expressions for each ODE in the system. We used a high-performance computing (HPC) cluster to estimate the model parameters in parallel, using the pymoo's implementation of differential evolution (Blank and Deb, 2020; Price et al., 2006). The hyperparameter settings for differential evolution were as follows: strategy=best1bin, mutation=0.5, cross-validation=0.5, maximum iterations=2000, population size=60. The objective function for the optimization algorithm was the root-mean-square error between observed (numerically calculated derivatives) and predicted (calculated by the expression) values of time derivatives of the state variables (see Sect. 3.1).

Once the parameters of all models are estimated on training sets, the 2500 expressions for each equation in the system are first evaluated on a validation set, consisting of trajectories from four sets of initial values. The model with the lowest average validation trajectory error is selected as the best model. Subsequently, this model is re-evaluated on the test set, also containing trajectories from four sets of initial values. The average error on the four test set evaluations was used as the final metric of reconstruction performance.

Similar to ProGED, we performed experiments with DSO on an HPC cluster, separately for each equation in the system. Most of the settings from the default configuration file of DSO were left untouched. We have, however, reduced the number of samples from 2 million to five hundred thousand and the maximum sequence length to 64, in order to keep the computation time manageable. We also customized the function set to each problem type. For the generic systems we used a function set with the *addition, subtraction, multiplication, division, constant, sine and cosine functions*, while for the state oscillator systems, we dropped the trigonometric functions. Lastly, we used the same function set, but without division, for the phase oscillator systems. DSO returns two outputs, one is a set of candidate expressions obtained using Pareto front analysis, that represents an optimal trade-off between expression complexity and the coefficient of determination (R^2). Additionally, DSO outputs a set of 100 best expressions based only on the R^2 ("Hall of Fame"). We combine the candidate models and evaluate them on the validation and test sets as described above for ProGED.

The evaluation of SINDy followed a similar approach as for ProGED and DSO. We grouped the systems into three types (state oscillators, phase oscillators and others) and defined the function sets accordingly. We present them in detail in the Appendix. In contrast to ProGED and DSO, which discover each ordinary differential equation (ODE) in a system separately, SINDy discovers the entire system at once. Furthermore, SINDy can only reconstruct ODEs that are linear in parameters, therefore, it is not able to reconstruct the *bacres*, *predprey*, and *cphase* systems, which are nonlinear. Nevertheless, we performed system identification on the data using SINDy by adding specific nonlinear terms directly in the library (see Appendix B in order to have a comparable evaluation between the methods).

We chose SR3 (Zheng et al., 2018) as the optimizer in SINDy, with the hyper-parameters set to L1 regularization, maximum iterations of 10^5 , and a tolerance of 10^{-5} . As we wanted to make the evaluation process for different methods as similar as possible, we varied the regularization parameters *threshold* and ν to obtain 400 candidate systems, which were used in the model selection procedure. The *threshold* parameter, which determines the strength of the regularization, varied between 0.1 and 10, with a 0.05 step, while ν , which determines the level of relaxation, was set to either 0.5 or 1. All 400 candidate models were evaluated on the validation and test sets as described for ProGED above.

Lastly, we attempted to use a transformer-based method for symbolic regression (Kamienny 2022) and compare it to other approaches. However, we obtained suboptimal results, that could have been caused by incomplete documentation rather than the method itself. Thus, we do not report the results for this method.

Our approach primarily centers on leveraging domain knowledge for the formulation of grammars. However, to facilitate the comparative analysis we also conducted *unconstrained* experiments, where system identification was performed without any prior assumptions. Essentially, we duplicated the experiments with the expanded space of potential models, which was done by increasing the set of possible functions or production rules. Concretely, for ProGED, we formulated the grammar as detailed in Table 9 in Appendix. To increase the search space for DSO and SINDy (see Appendix B), we additionally incorporated logarithmic and exponential functions in their function libraries.

4.4 Partial observability

In the partial observability scenario, we compared ProGED with two other methods, L-ODEfind and GPoM, which are capable of handling partially-observed dynamical systems, unlike DSO and SINDy. We chose to test on only the *vdp* model, which was best reconstructed under full observability, as the identification under partial observability is a challenging task. Moreover, the *vdp* model includes only terms with linearity in parameters and can thus be reconstructed by L-ODEfind and GPoM. To create a scenario with partial observability, we simply removed the time-series data for one of the two state variables, depending on which one was observed, resulting in incomplete knowledge of the system's dynamics. We tested the performance of each method by varying the level of data coarseness and noise. The particular settings of each method are outlined in the subsequent paragraphs.

ProGED generated the candidate models for the *vdp* system with the same state oscillator grammar (Table 7) used in the full-observability experiments. Again, we let ProGED generate 2500 models; however, in this case, the grammar formulated the entire system at once, already encompassing differential equations for both state variables. As explained in Sect. 3.2, to work under partial observability, the objective function in ProGED utilizes an initial value ODE solver, concretely LSODA (with the hyper-parameters set to $\text{rtol}=10\text{e-}3$, $\text{atol}=10\text{e-}3$). As only one of the initial values is observed, the other has to be, together with other model parameters, estimated by the optimization algorithm. All other settings for the optimization algorithm were the same. We used the same validation-test procedure as for the full observability scenario to select the best model among the 2500 candidates.

L-ODEfind's main setting is the maximum polynomial order, which was set to three. In the complete observability scenario, we established the highest degree of ODEs to be one, while in the partial observability scenario, it was set to two. L-ODEfind outputs the best model in a higher dimensional space, which cannot be transformed back to the first-order system of ODEs uniquely. However, since we used the known VDP model that includes the equation $\dot{x} = y$, we were able to convert the second-order equation to a system of first-order ODEs, allowing us to compare and evaluate the method with the others.

For GPoM, we used similar hyper-parameters as for L-ODEfind. We fixed the maximum polynomial order at three. For cases where both variables were observed, we kept the number of dimensions to be created from each input variable at one. Under partial observability, we set this hyper-parameter to two, in order to create an additional state variable by differentiating the observed variable. The ODE integration method used was LSODA with a maximum limit of 5120 integration steps.

4.5 Performance measures

We compared the results of the different methods with three metrics that quantify either the accuracy of the reconstruction or the complexity of the resulting expression. The primary metric that was used for model selection, was the *trajectory error*, calculated on train, validation and test sets. We calculate it, for a given state variable u , by using the relative-root-mean-square-error

$$TE_u = \sqrt{\frac{\sum_{i=1}^n (u(t_i) - \hat{u}(t_i))^2}{\sum_{i=1}^n (u(t_i) - \bar{u})^2}}, \quad (3)$$

where $u(t)$ and $\hat{u}(t)$ denote the simulated values of the state variable u at time point t , computed using the true model and the reconstructed model, respectively. The $\bar{u}$ is the mean value of $u(t)$ in the data obtained by simulating the true model. We define the total trajectory error as the sum over all the state variables u in the system of equations, $TE = \sum_u TE_u$.

Lower trajectory error means better reconstruction of systems' dynamics.

The second metric we used was the *normalized term difference* (TD), defined as the sum of the number of missing terms and the number of wrong terms in the reconstructed system, divided by the number of true terms:

$$TD_u = \frac{N_{u,\text{missing}} + N_{u,\text{wrong}}}{N_{u,\text{true}}}. \quad (4)$$

Here, $N_{u,\text{missing}}$ is the number of terms that are missing from the reconstructed differential equation for the state variable u in the system of equations. The number of wrong terms $N_{u,\text{wrong}}$ is the number of terms that are not present in the true differential equation and the $N_{u,\text{true}}$ is the number of true terms. We define the *term* equivalently to a summand, which is an individual part of the expression, separated by addition or subtraction. Again, the total TD was calculated as the sum over all the state variables u in the system of equations, $TD = \sum_u TD_u$. We calculated TD using a custom script that first expands expressions by multiplying out all of the products and then simplifies them using the Sympy library. Additionally, the constants are rounded to the third decimal place and only the terms with constant above 0.001 are kept. The decision to round to the third decimal place was influenced by the presence of a non-zero third decimal place parameter in the *predprey* system, specifically in its parameter value of 0.075. After the expressions were simplified, the parameter values were replaced by the symbol "C" in order to allow comparisons between the terms. Additionally, the symbol "C" was added wherever it was missing. For example, the expression $8.456x + y \sin(x)$ would be transformed to $Cx + Cy \sin(Cx)$. Lastly, we counted the number of terms in the original and reconstructed models, as well as the number of common terms, from which TD was calculated. Finding the number of common terms proved to be challenging for models involving trigonometric functions, as these functions can be expressed in multiple equivalent yet non-identical forms (e.g., $\cot(x) = \cos(x)/\sin(x)$). To overcome this issue, additional equivalent terms were added in a set, which was especially needed for trigonometric functions (e.g., $C \sin(Cx + C) == C \cos(Cx)$). In the end, we also manually inspected the results of the automatic TD calculation and made necessary corrections to the TD values. We manually corrected the TD results in 9/180 model expressions in the first experiment and 3/180 evaluations in the second. The TD metric is a useful measure of both the accuracy and complexity of a reconstructed system of equations. The reason to use TD as the second measure of performance is to identify potential over-fitting, which may be overlooked by TE since the latter tends to favour more complicated models that closely match particular trajectories. Ideally, TD should be zero, indicating a perfect match between the reconstructed and true model structures.

The third measure of the reconstruction success was the *normalized complexity* (NC), which is calculated as the number of nodes in the expression tree of a reconstructed equation divided by the number of nodes in the expression tree of a true system's equation:

$$NC_u = \frac{N_{u, \text{ nodes in reconstructed}}}{N_{u, \text{ nodes in true}}} . \quad (5)$$

The *NC* of a system of equations was considered as the sum of the NC_u of the individual equations of state variables u in the model. NC is best at 1, where the complexity of the true and reconstructed model are equal. We included NC to complement TD. The NC metric is commonly used in research due to its straightforward calculation. Nevertheless, TD provides a more informative measure of accurate reconstruction. Even if NC is equal to 1, TD can still take nonzero values, implying the existence of erroneous terms in the reconstruction models. However, when TD is equal to 0, NC should also be 1.

The performance of the methods was statistically evaluated in terms of their trajectory error for each system by using a critical difference diagram (Demšar 2006). The critical difference measure includes the Friedman test with corresponding post-hoc Wilcoxon tests for pair-wise comparison between the methods. Statistical results were corrected for multiple comparisons using Holm's method (Fawaz et al., 2019). We performed the statistical evaluation separately on six dependent configurations, consisting of pairs of the three SNR values and the two data types (small and large data). The statistical tests were applied in both, the constrained and unconstrained model space under full observability.

5 Results

The system identification performance of the methods for ODE discovery was evaluated by using three measures: the trajectory error (TE), which measures the success of reconstructing the dynamics of the systems, the normalized term difference (TD), which measures the success of reconstructing the model structure and normalized complexity (NC), which describes the relative complexity of the reconstructed model as compared to the complexity of the true model. We begin by presenting the results in the context of full observability, considering both constrained and unconstrained model spaces. Following that, the results for the partial observability setting are presented, where only the constrained model spaces were considered.

5.1 Full observability

5.1.1 Results of statistical analysis

To compare the three methods, we conducted a statistical analysis of their test TE results using critical difference diagrams. The analysis was conducted separately for each of the six configurations, which included combinations of two data types (*small* and *large* data) and three SNR levels (*clean*, *30 dB* and *13 dB SNR*). We provide the results of the statistical analysis in Table 3, where the ranks, p-values, and critical difference measures of the Friedman test are reported. The group-level Friedman test showed no significant differences when constrained model spaces were considered. In certain configurations, one method may demonstrate superiority, while in other configurations, different methods may

Table 3 Statistical comparison of methods based on their trajectory errors on test data

		Constrained		Unconstrained	
		Small	Large	Small	Large
Clean	ProGED	1.8	1.8	1.4	1.6
	DSO	2.0	2.0	1.9	2.0
	SINDy	2.2	2.2	2.7	2.4
	p-value	0.67	0.67	0.08	0.20
	CD	0.4	0.4	1.0	0.8
	N	10	10	10	10
30 dB	ProGED	2.1	1.80	1.4	1.75
	DSO	1.5	1.90	1.9	1.75
	SINDy	2.4	2.3	2.7	2.5
	p-value	0.12	0.50	*0.01	0.22
	CD	0.90	0.5	1.30	0.75
	N	10	10	10	8
13 dB	ProGED	1.80	2.13	1.67	1.29
	DSO	1.7	1.38	1.33	2.00
	SINDy	2.5	2.50	3.0	2.71
	p-value	0.15	0.07	*0.0009	*0.03
	CD	0.80	1.13	1.67	1.43
	N	8	10	9	7

Results are shown for both constrained and unconstrained model search spaces and for each of the six data type combinations resulting from two data sizes (small and large data sets) and noise levels (clean, 30 dB SNR and 13 dB SNR). The first three rows in each combination represent the ranks of each method, where rank 1 is the best, and rank 3 is the worst score. P-values based on the group-level Friedman test are shown in the fourth row, the critical difference (CD) in the fifth row and the number of samples (N) in the last row. The three combinations marked with a star * had significant group differences ($p < 0.05$). For these, the CD diagrams are shown in Appendix Fig. 5

exhibit better performance. The lack of significant differences can, in part, be attributed also to the small sample size. While the use of ten systems in six different configurations is an extensive evaluation, $N = 10$ is a small sample size for statistical tests. Additionally, SINDy was unable to construct and regularize valid models for some systems, which further impacted the statistical power.

Statistical analysis of the experiments with unconstrained model spaces also showed no statistical differences in performance between the methods on noise-free data. On the other hand, it showed significant differences in three noisy configurations: the large data at 13 dB SNR and the small data at 30 dB and at 13 dB SNR. In two out of three cases, SINDy had significantly lower rank (worse performance) compared to ProGED and DSO (see Appendix Fig. 5).

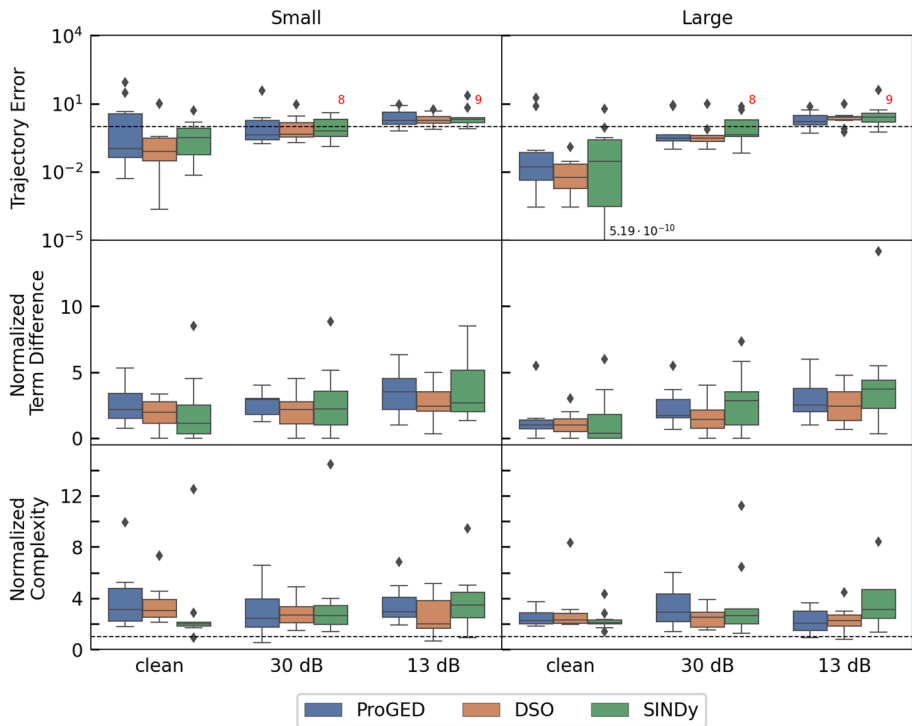


Fig. 1 Comparison of the trajectory error (top row), normalized term difference (middle row) and normalized complexity (bottom row) for the ProGED, DSO and SINDy methods, colour-coded as shown in the legend. The left-hand side plots correspond to results on the small data and the right-hand side plots to results on the large data. Each subplot compares performance on data with different noise/SNR levels: clean data, 30 dB and 13 dB. Each boxplot represents the distribution of performance over the ten dynamical systems. The black diamonds represent outliers. The red numbers denote the number of successful identifications when some out of the 10 system identifications failed (only in SINDy). Note that TE is shown on a logarithmic scale. The dashed horizontal line at $TE = 1$ represents the threshold, which can be seen as a minimum requirement for good performance. The line at $NC = 1$ represents the optimal complexity value. The optimal value for TD is 0. One value for SINDy was too small to fit on the plot, so the boxplot was truncated and the value is instead displayed numerically at the end of the whisker (Color figure online)

5.1.2 Descriptive analysis

A summary of the results of the experiments with constrained model spaces is shown in Fig. 1, where ProGED is compared to DSO and SINDy.

The results show that the reconstruction on clean, large data as expected outperformed all the other configurations (see the first group of three boxplots in the top right subplot). All three methods had a median TE of around 10^{-2} in this configuration, as well as the lowest TD, meaning that they made the least reconstruction mistakes. This is because noise-free, higher-resolution data is an ideal scenario, especially for methods that rely on numerical differentiation.

Both the presence of noise and the data size/granularity notably influenced the methods' performance. A comparison of the TE on the clean, small data with the TE on the moderately noisy (30 dB SNR) but large data reveals that moderate noise had a more pronounced effect on TE in all three methods. However, the TD and NC metrics present a somewhat different picture. There were more erroneous terms when provided with small, clean data

as compared to large, moderately noisy data. One plausible explanation for this discrepancy is that noise has a more substantial effect on parameter estimation, influencing TE more, as compared to the effect on the actual structure selection algorithm.

The assessments on highly noisy data (13 dB SNR) unsurprisingly yielded the poorest results, irrespective of data size/granularity. The TE was around or above 1 for all methods, and, on average, the reconstructed models contained more than twice the number of incorrect or missing terms compared to the terms in the true model.

As an additional observation, Fig. 1 nicely illustrates how the TD metric can offer additional insights alongside the NC metric. When examining how NC changes for each method in relation to noise levels, it demonstrates relative stability with some fluctuations. In contrast, the TD of each method monotonically increases with the level of noise, aligning with expectations and resembling the behavior of the TE metric. We have also computed the Pearson correlation coefficient ($r = 0.65$) between TD and NC, revealing a relatively strong correlation. All in all, the metrics capture similar aspects, but only to some extent, thus highlighting the importance of the term difference metric for evaluating the results of equation discovery.

A summary of the results obtained when the unconstrained model spaces were considered is given in Appendix C. Notably, SINDy exhibited significantly poorer performance and struggled to provide valid expressions in certain noisy scenarios. Even in the context of a large and clean dataset, the TD was approximately 5, signifying that the reconstructed expressions contained on average more than five times as many incorrect terms as the number of terms in the true expressions. We believe the reason for the worse performance can be the use of a large and considerably more general library (that does not include specific nonlinear terms that would be partially tailored to the Dynobench dataset), in contrast to the experiments with constrained model spaces. This made it difficult (and sometimes impossible) for linear regression to return parsimonious expressions, even with strong L1 regularization. Another notable observation is that DSO maintained consistent performance when assessed on a clean and large dataset. However, its performance declined when evaluated under alternative data configurations. A similar behavior was observed for ProGED, which exhibited poorer results across all configurations. The diminished performance of both DSO and ProGED can be attributed to the increased search space. Given

Table 4 Number of successful reconstructions of structure (TD = 0) and dynamics (TE < 0.1) for each of the six configurations (two data types and three noise levels)

		Constrained				Unconstrained			
		TD = 0		TE < 0.1		TD = 0		TE < 0.1	
		Sm	La	Sm	La	Sm	La	Sm	La
Clean	ProGED	4	10	12	18	2	6	8	13
	DSO	7	11	16	20	2	9	10	20
	SINDy	7	11	12	18	3	4	10	11
30 dB	ProGED	2	2	5	4	2	3	3	5
	DSO	6	6	4	6	2	4	2	5
	SINDy	3	4	3	5	2	3	1	5
13 dB	ProGED	1	2	0	0	2	2	0	0
	DSO	3	3	0	0	0	2	0	0
	SINDy	1	1	0	0	1	1	0	0

There were 21 expressions altogether (three ODEs in the lorenz model and two ODEs in the others)

ProGED's limitation in evaluating a finite number of candidate models, the search space expansion inevitably impacted its performance. This may not be the case for DSO, as it can better learn the structure when presented with a sufficient amount of good quality data.

Next, we inspected the number of successful reconstructions of model structure ($TD = 0$) and dynamics ($TE < 0.1$) for each of the six configurations. Table 4 shows the results on both the constrained and unconstrained model search spaces. Overall, the number of successful reconstructions decreases as noise increases, as well as between constrained and unconstrained search. If we first consider the constrained scenario, we observe that the methods at best find the true model structure in about half of the cases, when evaluated on clean and large data. ProGED successfully identified 10 out of 21 expressions, while both DSO and SINDy discovered 11/21. On the other hand, about twice as many models had good fit to the data ($TE < 0.1$). ProGED and SINDy found 18 such models and DSO found 20.

In the unconstrained search space scenario, as anticipated and consistent with other data configurations, the results show relatively lower performance. DSO identified a maximum of 9 structurally correct expressions, while ProGED and SINDy identified 6 and 4, respectively. Regarding expressions with good reconstruction of model dynamics ($TE < 0.1$), DSO led with 20, followed by ProGED with 13, and SINDy with 11.

Overall, DSO exhibited the most favorable results for both reconstruction success criteria. SINDy was most influenced by the type of library used and ProGED demonstrated competitive performance reconstruction of dynamics and a bit worse performance at structural reconstruction. This reflects its use of a potent parameter optimization algorithm and its disadvantage due to limiting the number of possible expression evaluations, impacting model selection performance.

In general, structure identification proved harder as compared to the identification of dynamics, as it is widely known in the field. The poorer structural results could be to some extent influenced by the relatively strict TD measure employed in our study. As detailed in Sect. 4, we retained only the terms with constants above 0.001. This stringent threshold for constant values might have in some cases led to the inclusion of irrelevant terms that had very little influence on the accuracy of the reconstructed model. Nevertheless, rounding to the third decimal place is generally considered to be a relatively lenient threshold. Furthermore, the contrast in success rates between structure identification and dynamics identification again underscores the importance of term difference evaluation. Within the field of system identification, the focus is often on achieving similar dynamics on the test data, a goal that is relatively easier to attain. However, relying solely on the resemblance of dynamics does not conclusively indicate the discovery of the correct underlying model.

The last results we present for the full observability scenario are the best-generated models for each of the ten tasks of dynamical system reconstruction. The best reconstruction was based on the TE metric, considered separately for each system, data configuration and SNR level. In Table 5, we provide the results for the large, clean data within the constrained search spaces. The corresponding results for the same data configuration within the unconstrained model spaces is given in the Appendix Table 10. The complete results across the other data configurations can be found in the GitHub repository.

Overall, ProGED and DSO found the best approximation of dynamics for three models, whereas SINDy demonstrated superior performance on four models. SINDy found the best models for glider, shearflow, barmag and stl systems. DSO was the best in reconstructing the vdp, lorenz and lv systems, while ProGED was the best in reconstructing bacres, cphase and predprey models. Noteworthy that while the best identification of the stl model by SINDy could be expected, the identification of other nonlinear models, which are typically challenging for linear regression methods, was unexpected. This can be attributed

Table 5 Best reconstructed models of ten dynamic systems based on trajectory error (TE). The results shown here are based on the fully observed setting, with constrained model spaces and clean, large data. The column "Best method" lists the method that reconstructed the model with the lowest TE, which is, together with term difference (TD), shown in the third column. The names of the systems are shown in the first column

System	Best method	TE, TD	Best model
bacres	ProGED	$3.4e-3$ 0.67	$\dot{x} = -\frac{19.98y}{9.786x+4.013} - 0.999x + 19.994$ $\dot{y} = -\frac{19.746xy}{9.873x^2-19.754} + 10.0$
barmag	SINDy	$5.2e-10$ 0	$\dot{x} = -1.0 \sin(x) + 0.5 \sin(x-y)$ $\dot{y} = -1.0 \sin(y) - 0.5 \sin(x-y)$
chpase	ProGED	$2.3e-2$ 0.75	$\dot{x} = 0.8 \sin(1.0x) + 0.8 \sin(1.0y) + 0.001 \cos(1.029y)$ $+ 1.728 \cos(0.003y + 4.621) + 2.144$ $\dot{y} = 0.6 \sin(1.0y) + 4.53$
glider	SINDy	$1.0e-3$ 0	$\dot{x} = -0.05x^2 - 1.0 \sin(y)$ $\dot{y} = 1.0x - 0.997 \frac{\cos(y)}{x}$
lorenz	DSO	$1.7e-3$ 0.5	$\dot{x} = -10.0x + 10.0y$ $\dot{y} = -1.0x(z - 26.998) + 1.0x - 1.0y$ $\dot{z} = xy - 2.667z + 0.009$
lv	DSO	$5.2e-3$ 0	$\dot{x} = x(-x - 1.995y + 3.0)$ $\dot{y} = 0.998y(-x - y + 2.0)$
vdp	DSO	$2.0e-4$ 0	$\dot{x} = y(1.0 - 1.0x)/(1.0 - 1.0x)$ $\dot{y} = -1.999x^2y - x + 1.999y$
predprey	ProGED	$1.1e-2$ 1	$\dot{x} = -0.999x^2 + 3.997x^2/(x + 0.001)$ $- 1.0xy/(x + 1.001) + 0.004$ $\dot{y} = 1.0xy/(x + 1.0) - 0.075y^2$
shearflow	SINDy	$1.6e-9$ 0	$\dot{x} = 1.0 \cos(x)/\tan(y)$ $\dot{y} = 0.9 \sin(x) \cos(y)^2 + 0.1 \sin(x)$
stl	SINDy	$3.4e-4$ 0.25	$\dot{x} = -1.0x^3 - 1.0xy^2 + 1.0x - 3.0y$ $\dot{y} = -0.998x^2y + 0.001xy^2 + 2.999x - 1.001y^3 + 1.0y$

to SINDy's custom function libraries (see Appendix B), which included not only the elementary functions (e.g. $\cos$, $\sin$) but also their combinations, such as $\cos^2(y) \sin(x)$ - direct terms in the true models. This provides an important advantage for SINDy, as it only needs to retain these terms, while ProGED and DSO have to construct the correct terms from elementary functions. While we extended SINDy's library with high-level terms for the sake of comparison, the inclusion of such specific terms may have limited applicability in real-world scenarios. This is further confirmed by results with the more general, unconstrained library, as shown by the results in the Appendix Table 10. In that case, SINDy was only able to best reconstruct the polynomial stl model. The unconstrained model search space posed challenges for ProGED as well. ProGED managed to best reconstruct 3 out of 10 models, but with higher TD and TE values. As ProGED's strength lies in formulating expressive grammars to narrow down the search spaces, the results on unconstrained space can be seen as a baseline assessment for evaluating ProGED's performance.

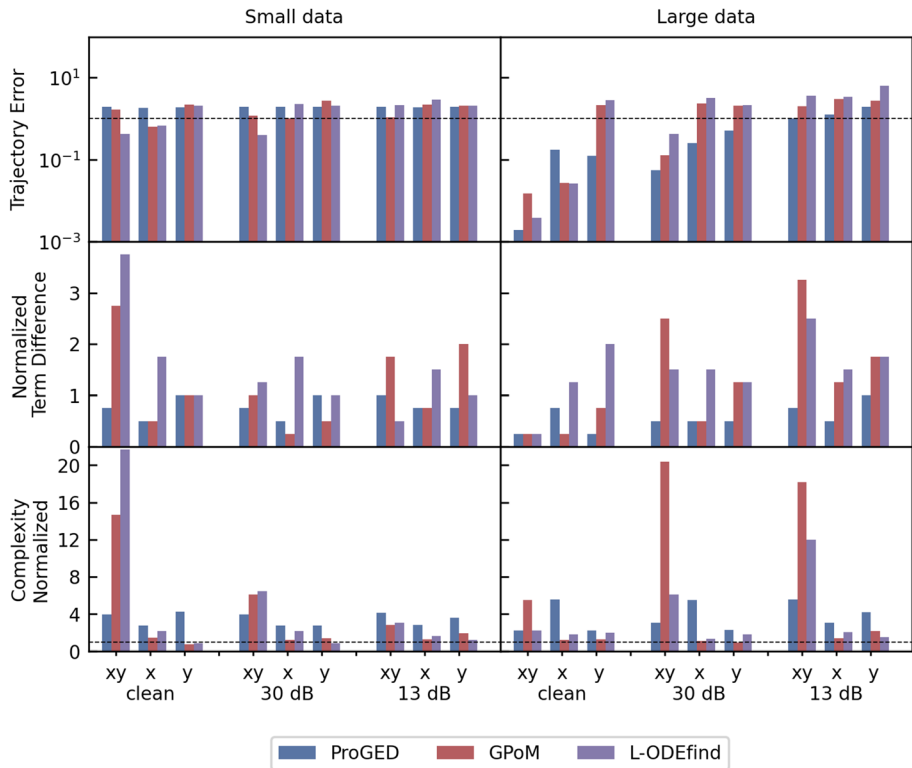


Fig. 2 Comparison of the performance of three ODE discovery methods under partial observability, with each method colour-coded according to the legend. The top row displays the trajectory error (TE) on a logarithmic scale, the middle row the term difference (TD) and the bottom row the normalized complexity (NC). The graphs are arranged into two columns based on data length, as indicated by the subtitles. On the x-axes, the results are hierarchically grouped by SNR (clean, 30 dB, 13 dB) and by the observability modes: full observability (labelled as xy), observability in only the x variable (labelled as x), and observability in only the y variable (labelled as y). Since only one system (vdp) was tested in the partially observed setting, error bars are not included. The dashed horizontal line at $TE = 1$ represents the threshold/ minimum requirement for good performance, while the line at $NC = 1$ represents the optimal complexity value. The optimal value for TD is 0 (Color figure online)

5.2 Partial observability

To evaluate the performance of ProGED in identifying partially observed systems, we followed a similar procedure as with fully observable systems. We considered six different data configurations, including two types of data (small and large) and three SNR levels (clean, 30 dB, and 13 dB). The evaluation process consisted of employing the same validation-test procedure and comparing the results of ProGED with two other methods. The latter are no longer DSO and SINDy, which can only deal with fully observable systems, but rather L-ODEfind and GPoM, that were designed to handle partial observability. However, instead of evaluating on the entire Dynobench dataset, we focused on the reconstruction performance for the vdp model, considering only the constrained model space.

Figure 2 presents the results in terms of three performance measures: TE, TD and NC. The first thing to notice in the TE plots (top row) is that a sufficient amount of data were necessary for good performance. On the small data either the reconstruction of dynamics was poor ($TE \approx 1$) or the TD and NC of the models were large. The one exception is the reconstruction by GPoM on data with 30 dB SNR and only the variable x observed, where the method added only one erroneous term to the otherwise correct model ($TD = 1$). Notably, in the context of partial observability, it appears that the amount of data has a significantly greater influence on TE than the level of noise in the data, which was not as clear under full observability. One potential explanation for this is that both ProGED and GPoM employ ODE simulation, which is less sensitive to noise in the data as opposed to numerical differentiation approaches.

Due to the poor results on small data, our examination of performance under different observability scenarios (columns xy , x and y) focuses only on the results from the large dataset. As expected, the reconstruction performance under partial observability decreases with decreasing SNR. On clean, large data, all the methods showed relatively good reconstruction of dynamics under at least one of the two partial observability settings. However, only ProGED was able to reconstruct vdp under the moderate SNR of 30 dB. Interestingly, while ProGED performed similarly under both partial observability settings

Table 6 Best models of vdp system reconstructed in terms of trajectory error (TE) in the partially observed setting

SNR	Obs.	Best method	TE, TD	Best model
Clean (Long data)	xy	ProGED	0.002	$\dot{x} = 1.0y$
			1	$\dot{y} = 0.001x^4y - 2.004x^2y - 1.0x + 2.002y$
	x	LODefind	0.027	$\dot{x} = y$
			5	$\dot{y} = -0.001x^3 - 1.993x^2y - 0.002xy^2 + 0.003xy$ $-0.994x - 0.001y^2 + 1.997y + 0.003$
	y	ProGED	0.124	$\dot{x} = 0.984y$
			1	$\dot{y} = 0.001x^4y - 2.066x^2y - 1.016x + 2.001y$
30 dB (Long data)	xy	ProGED	0.057	$\dot{x} = 1.005y + 0.005$
			2	$\dot{y} = -1.963x^2y - 0.993x + 1.988y - 0.01$
	x	ProGED	0.257	$\dot{x} = 0.004x^2 + 0.865y - 0.013$
			2	$\dot{y} = -1.984x^2y - 1.15x + 1.974y$
	y	ProGED	0.507	$\dot{x} = 0.938y$
			2	$\dot{y} = 0.147x^4y - 2.695x^2y - 1.075x + 2.134y$ $+0.002$
13 dB (Long data)	xy	ProGED	1.021	$\dot{x} = 0.064x^2 + 0.968y - 0.129$
			3	$\dot{y} = -2.328x^2y - 1.081x + 2.071y - 0.16$
	x	ProGED	1.259	$\dot{x} = 0.651y - 0.022$
			2	$\dot{y} = -1.985x^2y - 1.527x + 1.983y + 0.064$
	y	ProGED	1.973	$\dot{x} = 0.156x + 1.098y$
			4	$\dot{y} = \frac{2.872x^3}{x+4.804} - 4.972x - 4.773y - 0.146$

The results are based only on the "large" data that was either clean, or had noise with 30 dB and 13 dB SNR. Additionally, the results are separated by observability mode, where xy represents the full observability scenario, while x and y denote partial observability (of only one respective state variable). The column "Best method" lists the method that reconstructed the model with the lowest TE

(when either x or y were observed), GPoM and L-ODEfind showed a bigger discrepancy and performed well when only x was observed, but not when only y was observed. One reason for that could be that L-ODEfind constructs a model in a higher dimensional space. An example of the returned model by L-ODEfind, when only x is observed, is $\ddot{x} = -0.94 + 1.75\dot{x} - 1.84\ddot{x}x^2 - 0.10\ddot{x}^2 + 0.31\ddot{x}^3$. The transformation back to the original model is only possible because we are dealing with the vdp system, where $\dot{x} = y$, otherwise it would not be possible. The L-ODEfind is consequently not useful for system identification but rather for the prediction of the trajectory of the observed state variable. GPoM, on the other hand, returns a system of first-order equations, although it still works in a higher dimensional space to find the correct model. Upon examining the TD values presented in the second row, it is evident that none of the methods achieved a perfect reconstruction of the model structure for the vdp system: each had at least one additional erroneous term in the model, also under full observability (scenario xy).

Next, similarly to the analysis under full observability, we looked at the best-reconstructed models for each of the data configurations and summarized the results in Table 6. ProGED demonstrated superior performance across all scenarios, except in the case of clean data with only the x variable observed, where L-ODEfind was better. These results

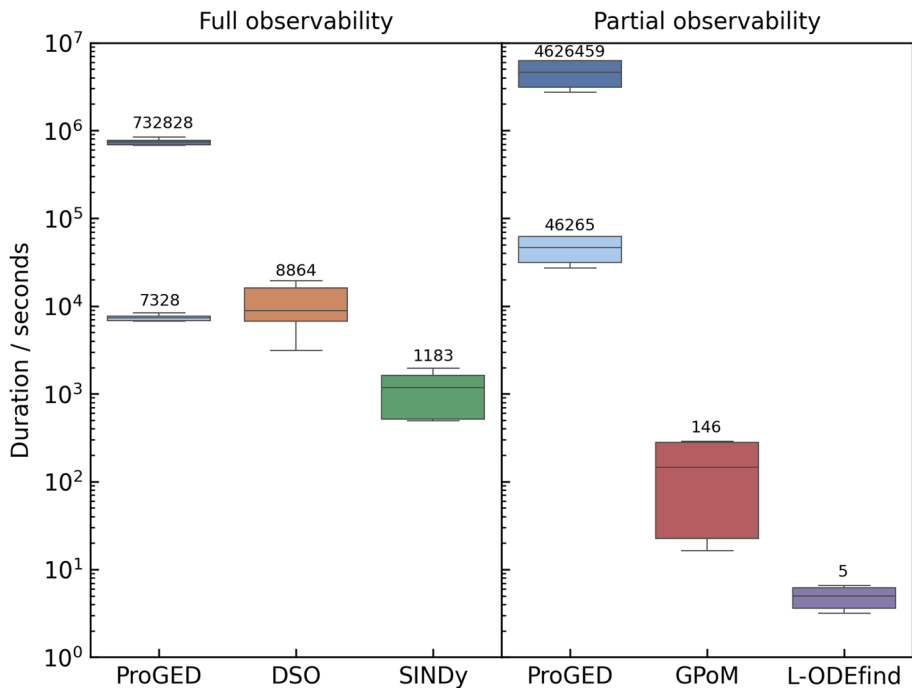


Fig. 3 A comparison of system identification duration for the evaluated methods, separately for full observability (left graph) and partial observability (right graph). Each boxplot depicts the distribution of duration for a specific method across the systems used and across all configurations of data length and SNR levels. Median values are explicitly annotated on the boxplots, providing precise numerical information. Notably, the duration of ProGED duration is represented by two boxplots. The dark blue boxplot indicates the summed duration needed for the system identification, while the light blue boxplot represents the duration needed on the computer cluster, where the model evaluations were done in parallel. Note that the y-axis is in logarithmic scale (Color figure online)

also nicely highlight the presence of additional erroneous terms in the models, as mentioned in the previous paragraph. Additional terms have minimal impact in the simpler configurations (higher SNR and larger data), but become more influential in more challenging data configurations.

5.3 The time complexity of experiments

In our experiments, the computation time required for ODE discovery under each configuration varied depending on the method, data size, SNR levels, observability, as well as the number of model parameters of the individual systems in the benchmark.

Figure 3 depicts the duration of inferring a single dynamical system for each of the evaluated methods, where we added up the running times of the four individual datasets (obtained from four different initial states) and the two state variables (or three for lorenz system). Each boxplot shows the distribution of duration over all six configurations of SNR levels and data coarseness/sizes. The computational costs are shown separately for full observability (left) and partial observability (right).

SINDy, GPoM, and L-ODEfind, which are capable of constructing and assessing only models that are linear in their parameters, exhibit shorter inference times as compared to ProGED and DSO, which can infer nonlinear models as well. The observed difference in computational costs can be attributed to several factors. Firstly, nonlinear models are inherently more complex and often involve a greater number of parameters. Additionally, the inference process for nonlinear models necessitates the utilization of computationally expensive optimization algorithms, both for structure as well as parameter search.

Note that Fig. 3 shows two boxplots for ProGED in each setting. For the full observability setting, the upper dark blue boxplot with a median duration of 732828 seconds (8.5 days) depicts the overall time ProGED would need if all of the 2500 models were evaluated sequentially. However, due to independent model sampling from the grammar, ProGED can estimate the parameters of the models independently in parallel. This improves the speed by the number of batches, which was 100 in our case. The duration of the experiments ran in parallel is then represented by the lower light blue boxplot with a median duration of 7328 seconds (2 hours).

While not depicted in the plot, both ProGED and DSO were additionally eight-fold accelerated through the parallel execution of system identification on each dataset (for the four initial values) and each of the state variable separately.

ProGED experiments were conducted on a high-performance computing cluster (HPC). For a single dynamical system, we tested 6 data configurations (two versions of data size and three noise levels), each on four sets of initial values. Additionally, 2500 candidate models were tested for each state variable in the system, with an evaluation process that was divided into 100 batches. Overall, this resulted in a total of $24 \cdot 2 \cdot 100 = 4,800$ performed jobs per system (for the lorenz system it was 7200 jobs, as it has three state variables). In the experiments involving partial observability, system identification was similarly done in parallel by splitting the jobs into the standard 24 data configurations, for each of the observability scenarios. Each of the configurations was run in 250 batches yielding $24 \cdot 3 \cdot 250 = 18,000$ jobs in total.

The DSO experiments included 504 jobs and involved the same data configurations as ProGED under full observability. DSO however, does not estimate the expressions in parallel. Thus, the model search for a particular configuration was computed within a single job.

SINDy is in general very fast and can complete system identification in several seconds. In our approach, SINDy conducted 96,000 local runs, resulting from varying 10 systems, 24 ($2 \cdot 3 \cdot 4$) data configurations and 400 different sets of regularization parameters. We did not have to separate the discovery of each state variable separately when evaluating SINDy, as it can find equations of all state variables simultaneously.

As shown by Fig. 3, L-ODEfind needed on average 5 seconds for one system identification and there were no additional runs. GPoM needed on average 145 seconds for one system identification. Both methods had to be run on 72 data configurations as described for ProGED. SINDy, L-ODEfind and GPoM were all run on a single computer.

6 Discussion

In this work, we presented an extension of ProGED, an equation discovery method based on probabilistic context-free grammars, in the direction of discovering ordinary differential equations (ODEs). This extension can additionally learn ODEs from partial observations, where only a subset of the state variables of the dynamical system are measured. We have also presented a systematic and comprehensive comparative evaluation of ProGED and different competing methods for modeling dynamical systems on an extensive new benchmark of ten dynamical systems.

The comparative evaluation involved the reconstruction of known ODEs from given data of various levels of coarseness, noise, and observability. This allowed the assessment of the methods' robustness, as well as their strengths and limitations related to their applicability to practical tasks of modeling dynamical systems from data. The methods were tested on an updated and extended version of the Strogatz benchmark that consists of ten dynamical systems, which we have also made openly available under the name Dynobench. The main differences between the Dynobench and Strogatz benchmarks are (i) the addition of three dynamical systems, expanding the variability of the models, (ii) the addition of six data configurations, encompassing two distinct data coarseness levels and three signal-to-noise ratio (SNR) levels, and last but not least, (iii) the use of a more accurate ODE solver to generate the data trajectories, better exemplifying the dynamics of the model. These adjustments allowed for a more comprehensive and realistic evaluation of the methods.

Looking at the effects of data coarseness, all the methods performed (more) poorly when applied to a small/coarse dataset containing only 100 data samples, sampled at 0.1 Hz. Conversely, as anticipated, the reconstruction of model dynamics improved significantly when the methods were applied to larger/finer datasets (2000 data points, sampled at 0.01 Hz). We highlight this point given the common usage of benchmarks which include a limited number of data samples (La Cava et al., 2021). As science increasingly leans towards gathering more high-quality data, the performance of ODE discovery methods should also be tested on such data. The length and the sampling rate of the large data we used are still well within the range of common sets of measurements in real-world experiments.

As expected, the results of the experiments confirm the well-established concept that the measurement noise greatly affects the ability to identify an underlying dynamical system (Fajardo-Fontiveros et al., 2023). The evaluated methods had great difficulties when the signal-to-noise ratio (SNR) was the lowest. Interestingly, under full observability and when the SNR was moderate, noise had relatively more influence on the results than the data size and coarseness. Under partial observability, data coarseness had a greater influence on the results as compared to the influence of SNR. This can be to some degree explained by the fact that

the methods employing numerical differentiation (normally used under full observability) would demonstrate a higher drop in performance when exposed to noisy data, as compared to the methods that use numerical simulation of ODEs (which is necessary under partial observability).

In the full observability setting, there were no statistically significant differences in performance observed among the methods, although DSO exhibited the best performance, followed by ProGED and then SINDy. Notably, despite the reasonable quality of the reconstructed model dynamics, the methods correctly identified the true model structure only for half of the equations. To evaluate the success of model structure identification, we employed both the term difference and the normalized complexity measures. Although the (normalized) complexity measure is often employed in the literature, due to its simplicity of calculation (Mundhenk et al., 2021; Kamienny (2022), we show that the term difference provides a more accurate assessment of the correctness of the model structure. While some studies already assessed the accuracy of model structure reconstruction using metrics similar to term difference (Mangiarotti and Huc 2019), we believe there is a need to establish a standard evaluation measure for the success of model structure reconstruction.

ProGED outperformed the other two methods, L-ODEfind and GPoM, in the partial observability setting and was the only one that completely successfully reconstructed the *vdp* system in settings with moderate noise and with only one of the state variables observed. Moreover, ProGED showed higher robustness in performance under partial observability (with different state variables observed). This property is essential in real modeling scenarios, where we cannot choose the measurement variables freely.

Lastly, while methods that rely on the polynomial structure (GPoM) or linearity in parameters (SINDy, L-ODEfind) perform fast and sufficiently well for some systems, ProGED and DSO are able to generate and identify much larger spaces of possible ODE models. This comes at the cost of longer running times, which seem to be, at least up to now, unavoidable. The response time of reconstruction experiments can be greatly reduced through extensive parallel implementation, particularly in the case of ProGED. Alternatively, the speed of ProGED can be further increased by using more restricted grammars, but this is only advisable in practical applications where domain knowledge is available.

The research presented here has some limitations that need to be addressed in further work. Firstly, although the construction of grammars based on background knowledge showed promise, there is room for improvement. In future research, optimization algorithms could be employed to optimize the production probabilities within the grammar in a data-driven manner. Another strategy to consider is the utilization of attribute grammars to provide additional constraints and refine the search space (Brence et al., 2023). Furthermore, exploring alternative generative algorithms (Mežnar et al., 2023), instead of relying solely on the Monte Carlo algorithm, could enhance the efficiency of ProGED's structure search. Secondly, the computation of the term difference (TD) measure, which we advocated as a means to evaluate the accuracy of the model structure, has implementation challenges. This is primarily due to the potential abundance of equivalent, but not identical expressions that can be generated by the modeling algorithms. Consequently, a manual inspection of the term difference measure was conducted, which, although subjective, was deemed necessary from our perspective. Lastly, increasing the number of systems and systems' trajectories used for the evaluation would provide more reliable evaluations and higher statistical power (La Cava et al., 2021). This would be especially informative under partial observability.

In sum, we show that the enhanced version of ProGED that we propose here can be successfully used to model dynamic systems. It exhibits comparable success rates to other contemporary methods for ODE discovery, when dealing with fully observable systems. It

is the preferred choice when dealing with partially observable systems and data with moderate levels of measurement noise.

Appendix

A Grammars for generating ODEs by ProGED

See Tables 7, 8, 9.

Table 7 A grammar designed to construct mathematical expressions on the right-hand side of the ordinary differential equations of the *state* oscillators models

$\mathcal{N}$	$= \{E, T, M, V\}$	
$\mathcal{T}$	$= \{+, *, \div, (,), c, x, y\}$	
$\mathcal{R}$	$ \begin{aligned} &= \\ &E \rightarrow E + T \ [0.6] && T \ [0.4] \\ &T \rightarrow T * V \ [0.35] && M * V \ [0.1] \ c \ [0.55] \\ &M \rightarrow x \div (x + c) \ [0.5] && y \div (y + c) \ [0.5] \\ &V \rightarrow x \ [0.5] && y \ [0.5] \end{aligned} $	
S	$= E$	

This grammar can generate expressions (E) involving sums (S) of multiplicative terms (T) involving state variables (V) and a Monod transformation thereof, M . The example grammar involves two state variables, x and y . It can be easily extended to an arbitrary number of state variables by adding new production rules to the nonterminals V and M . The number in square brackets next to each production rule denotes the rule probability

Table 8 A grammar designed to construct mathematical expressions on the right-hand side of the ordinary differential equations of the *phase* oscillator models

$\mathcal{N}$	$= \{E, T, L, LT, V\}$	
$\mathcal{T}$	$= \{+, *, \sin, \cos, (,), c, x, y\}$	
$\mathcal{R}$	$ \begin{aligned} &= \\ &E \rightarrow E + T \ [0.6] && T \ [0.4] \\ &T \rightarrow c * \sin(L) \ [0.5] && c * \cos(L) \ [0.5] \\ &L \rightarrow L + LT \ [0.1] && LT \ [0.9] \\ < \rightarrow c * V \ [0.2] && c \ [0.8] \\ &V \rightarrow x \ [0.5] && y \ [0.5] \end{aligned} $	
S	$= E$	

This grammar can generate expressions (E) involving linear combinations of trigonometric terms (T), each being a sine or a cosine of a linear combination (L) of the state variables (V). The example grammar involves two state variables, x and y . It can be easily extended to an arbitrary number of state variables by adding new production rules to the nonterminal V . The number in square brackets next to each production rule denotes the rule probability

Table 9 A universal grammar designed to construct mathematical expressions, without any domain knowledge

$\mathcal{N}$	$= \{E, F, T, R, V\}$		
$\mathcal{T}$	$= \{+, *, /, \sin, \cos, \log, \exp(,), c, x, y\}$		
$\mathcal{R}$	$=$		
$E \rightarrow E + F$ [0.6]	$ F$ [0.4]		
$F \rightarrow F * T$ [0.2]	$ F/T$ [0.2]	$ T$ [0.6]	
$T \rightarrow R$ [0.2]	$ V$ [0.4]	$ c$ [0.4]	
$R \rightarrow (E)$ [0.6]	$ \sin(E)$ [0.1]	$ \cos(E)$ [0.1]	$ \log(E)$ [0.1]
$V \rightarrow x$ [0.5]	$ y$ [0.5]		
S	$= E$		

It can generate expressions (E) involving polynomial terms, as well as trigonometric, logarithmic and exponential functions of the state variables (V) or expressions thereof (E). The grammar involves two state variables, x and y . It can be easily extended to an arbitrary number of state variables by adding new production rules to the nonterminal V . The number in square brackets next to each production rule denotes the rule probability

B Function library of SINDy

The SINDy library to construct state oscillators, namely the lv, vdp, stl, lorenz and prey systems, as well as the bacres system, included eight functions: $x, y, x^2, y^2, xy, x^3, x^2y$ and y^2x .

The library to construct phase oscillators (for the bargmag and cphase systems) included ten functions: $\sin(x), \sin(y), \cos(x), \cos(y), \sin(x + y), \cos(x + y), \sin(x - y), \cos(x - y), \sin(y - x), \cos(y - x)$.

The function library for the glider and shearflow systems included 31 terms: $x, y, xy, x^2, y^2, x^3, y^3, \frac{x}{y}, \frac{y}{x}, x^2y, y^2x, \sin(x), \sin(y), \cos(x), \cos(y), \tan(x), \tan(y), \cot(x), \cot(y), \frac{\cos(y)}{x}, \frac{\sin(y)}{x}, \frac{\cos(x)}{y}, \frac{\sin(x)}{y}, \sin^2(y) \sin(x), \sin^2(x) \sin(y), \cos^2(x) \sin(y), \cos^2(y) \sin(x), \sin(x) \cot(y), \sin(y) \cot(x), \cos(x) \cot(y), \cos(y) \cot(x)$.

The general function library that was in the experiments with the so-called unconstrained model search space included polynomial terms up to third order, Fourier library of sine and cosine functions up to first frequency, as well as custom functions $\frac{x}{y}, \frac{y}{x}, \tan(x), \tan(y), 1/\tan(x), 1/\tan(y), \log(x), \log(y), \exp(x)$ and $\exp(y)$, which altogether included 25 terms. For the lorenz system, the additional state variable z was included.

C Additional system identification results for unconstrained model search spaces

Additional results of system identification for unconstrained model search spaces are given in Fig. 4 and Table 10.

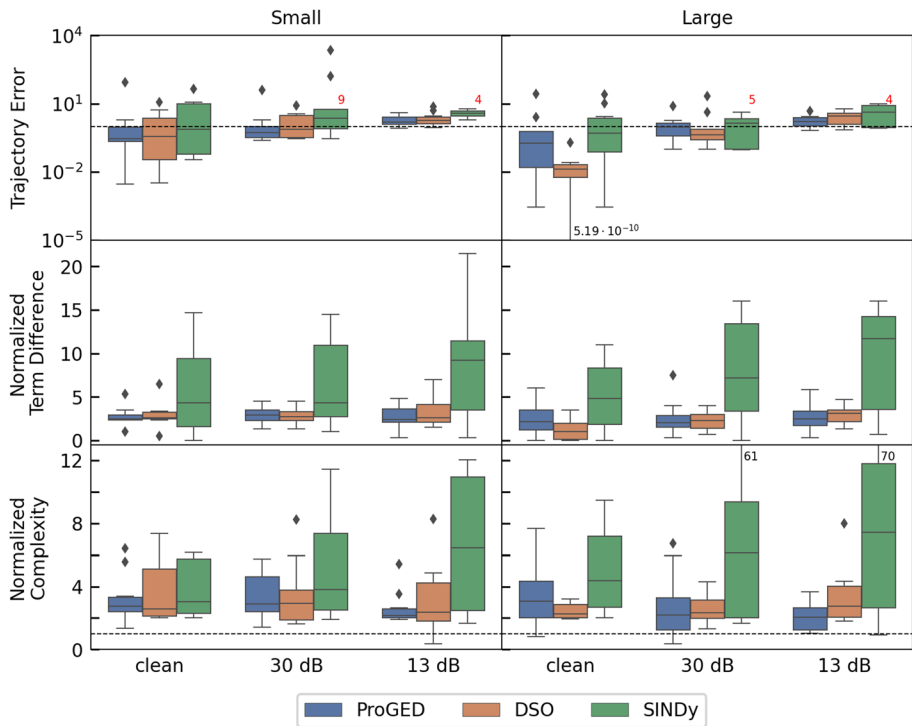


Fig. 4 Comparison of the trajectory error (top row), term difference (middle row) and normalized complexity (bottom row) for the ProGED, DSO and SINDy methods, colour-coded as shown in the legend. The left-hand side plots correspond to results on the small data and the plots on the right-hand side to results on the large data. Each subplot compares performance on data with different noise/SNR levels: clean data, 30 dB and 13 dB. Each boxplot represents the distribution of performance over the ten dynamical systems. The black diamonds represent outliers. The red numbers denote the number of successful identifications when some out of the 10 failed (important only for SINDy). Note that TE is shown on a logarithmic scale. The dashed horizontal line at $TE = 1$ represents the threshold, a minimum requirement for good performance. The line at $NC = 1$ represents the optimal complexity value. The optimal value for TD is 0. Some values for SINDy were too large or too small to fit on the plot, so the boxplots were truncated and the values are instead displayed at the ends of the whiskers (Color figure online)

Table 10 The best models reconstructed in terms of trajectory error (TE), for each of the ten systems in the fully observed setting, within the unconstrained model spaces. The results shown here are for the large, clean data configuration. The column "Best method" lists the method that reconstructed the model with the lowest TE, which is, together with term difference (TD), shown in the third column. The names of the systems are shown in the first column

System	Best method	TE, TD	Best model
bacres	ProGED	$1.2e-3$ 3.5	$\dot{x} = -0.997x + 19.335y/(-9.473x - 3.937)$ $-0.011 \log(x + 8.434) + 19.998$ $\dot{y} = 10.107 - (2.046y \exp((-0.741x - 0.399y)/(xy))$ $+ 2.046y \exp(1.002/y - (xy + 12.631x)/x^2)/x$ $- 8.634/x)/x - 3.083/x$
barmag	DSO	$5.2e-10$ 0	$\dot{x} = -1.0 \sin(x) + 0.5 \sin(x - y)$ $\dot{y} = -1.0 \sin(y) - 0.5 \sin(x - y)$
chpase	ProGED	$1.4e-2$ 1	$\dot{x} = 0.006y + 0.994 \sin(x) - 0.869 \sin(2.843$ $- \cos(0.174y^2 + 2.513)/y) + 2.248$ $\dot{y} = 0.6 \sin(y) + 4.53$
glider	DSO	$1.1e-2$ 3.5	$\dot{x} = (1.012 \sin(y) - 13.196) \sin(0.086x + 4.716)$ $- 13.193 \sin(y)$ $\dot{y} = x + 0.998 \cos(0.001x + y + 3.141)/x$
lorenz	DSO	$3.8e-3$ 0.5	$\dot{x} = -10.0x + 10.0y$ $\dot{y} = -x(0.993z - 26.759) + x - 0.948y$ $\dot{z} = xy - 2.667z + 0.009$
lv	DSO	$1.4e-2$ 0	$\dot{x} = x(-x - 1.98y + 2.998)$ $\dot{y} = y(-x - 1.0y + 2.0)$
vdp	ProGED	$2.9e-4$ 0	$\dot{x} = y(1.0 + 0.585/y) - 0.585$ $\dot{y} = -1.999x^2y - 1.0x + 1.999y$
predprey	DSO	$1.2e-2$ 1.83	$\dot{x} = -0.998x(x + y/(x + 0.999) - 4.001)$ $\dot{y} = y(-0.075y + 1.0 - 0.483/(0.483x + 0.483))$
shearflow	DSO	$2.0e-2$ 2	$\dot{x} = -0.998 \sin(y - 1.574) \cos(x)/\sin(y)$ $\dot{y} = (0.45 \sin(2.001y + 1.572) + 0.55) \sin(x)$
stl	SINDy	$3.8e-3$ 1.75	$\dot{x} = -0.866x^3 - 1.025xy^2 - 3.006y$ $+ 1.03 \sin(x) + 0.008 \sin(y)$ $\dot{y} = -1.017x^2y - 0.005xy^2 + 2.986x - 0.859y^3$ $+ 0.016 \sin(x) + 1.021 \sin(y)$

D Statistical analysis

The results of the statistical comparison of the performance between the three methods ProGED, DSO and SINDy are given in Table 3 (Sect. 5.2) and Fig. 5 (here).

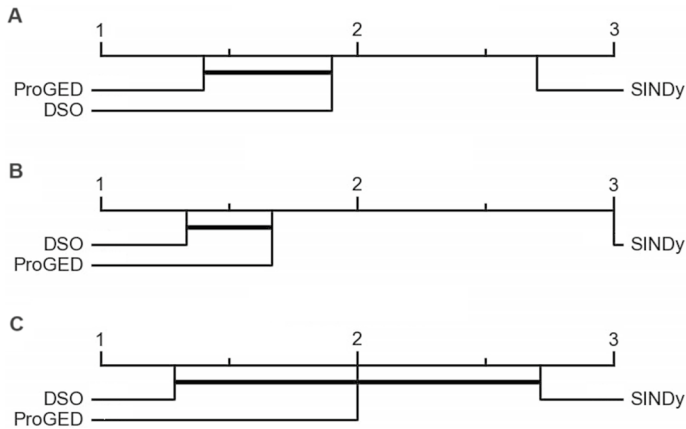


Fig. 5 The CD diagrams above graphically accompany Table 3. The diagrams show the rankings that had significant differences at the group level (p -level < 0.05). They occur in the unconstrained model search spaces setting, in the configurations with small data and moderate levels of noise (diagram A), small data and high level of noise (diagram B) and large data with high level of noise (diagram C)

Acknowledgements The authors thank Sebastian Mežnar for participating in discussions and providing valuable comments and feedback.

Author contributions Conceptualization: JB, LT, NO, BG, SD; Methodology: JB, LT, NO, BG, SD; Writing - original draft preparation: JB, LT, NO, BG; Writing - review and editing: JB, LT, NO, BG, SD; Funding acquisition: SD; Resources: SD; Supervision: LT, SD; Software: NO, JB, BG; Visualization: NO; Data curation: NO; Investigation: NO, BG; Validation: NO, BG.

Funding The authors acknowledge the financial support of the Slovenian Research Agency via the Research Program Knowledge Technologies (grant P2-0103) and the projects J1-3033, J2-2505, J2-4452, J2-4460, J3-3070, J4-3095, J5-4575, J7-4636, J7-4637, and N2-0236. It was also supported by the EC via the projects ASSAS (grant number 101059682), ELIAS (grant 101120237), INQUIRE (grant 101057499), PARC (grant 101057014), and TAILOR (grant 952215).

Data availability The datasets used in the experiments are available on Zenodo: <https://zenodo.org/records/10041312>.

Code availability All methods used in the experiments are available online, in the respective GitHub repositories. Additionally, we provide our code by which we obtained the presented results. It is located on the Github repository: [symreg_methods_comparison](https://github.com/symreg_methods_comparison).

Declarations

Conflict of interest The authors declare no conflicts of interest.

Ethical approval Not applicable.

Consent to participation Not applicable.

Consent for publication Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article

are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Blank, J., & Deb, K. (2020). pymoo: Multi-objective optimization in python. *IEEE Access*, 8, 89497–89509.
- Brence, J., Džeroski, S., & Todorovski, L. (2023). Dimensionally-consistent equation discovery through probabilistic attribute grammars. *Information Sciences*, 632, 742–756.
- Brence, J., Todorovski, L., & Džeroski, S. (2021). Probabilistic grammars for equation discovery. *Knowledge-Based Systems*, 224, 107077.
- Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15), 3932–3937.
- Čerepnalkoski, D. (2013) Process-based models of dynamical systems: Representation and induction. Ph.D. thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia.
- de Silva, B. M., Champion, K., Quade, M., Loiseau, J. C., Kutz, J. N., & Brunton, S. L. (2020). Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data. *Journal of Open Source Software*, 5(49), 2104. <https://doi.org/10.21105/joss.02104>
- Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7, 1–30.
- Džeroski, S., & Todorovski, L. (1993). Discovering dynamics. In *Proc. Tenth International Conference on Machine Learning* (pp. 97–103). San Mateo, CA: Morgan Kaufmann.
- Fajardo-Fontiveros, O., Reichardt, I., De Los Ríos, H. R., Duch, J., Sales-Pardo, M., & Guimerà, R. (2023). Fundamental limits to learning closed-form mathematical models from data. *Nature Communications*, 14(1), 1043.
- Hindmarsh, A.C. (1983) Odepack, a systemized collection of ode solvers. Scientific Computing.
- Hindmarsh, A. C., Brown, P. N., Grant, K. E., Lee, S. L., Serban, R., Shumaker, D. E., & Woodward, C. S. (2005). SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3), 363–396.
- Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, 33(4), 917–963.
- Kamienny, P.A., d'Ascoli, S., Lample, G., Charton, F. (2022). End-to-end symbolic regression with transformers.
- Kuznetsov, Y.A., Kuznetsov, I.A., Kuznetsov, Y. (1998). Elements of applied bifurcation theory, Vol. 112. Springer.
- La Cava, W., Orzechowski, P., Burlacu, B., de França, F.O., Virgolin, M., Jin, Y., Kommenda, M., Moore, J.H. (2021). Contemporary symbolic regression methods and their relative performance. arXiv preprint [arXiv:2107.14351](https://arxiv.org/abs/2107.14351)
- Mangiarotti, S., Coudret, R., Drapeau, L., & Jarlan, L. (2012). Polynomial search and global modeling: Two algorithms for modeling chaos. *Physical Review E*, 86, 046205.
- Mangiarotti, S., Huc, M. (2019). Can the original equations of a dynamical system be retrieved from observational time series? Chaos: An Interdisciplinary Journal of Nonlinear Science 29(2).
- Mežnar, S., Džeroski, S., & Todorovski, L. (2023). Efficient generator of mathematical expressions for symbolic regression. *Machine Learning*, 112, 4563–4596. <https://doi.org/10.1007/s10994-023-06400-2>
- Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K., Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., & Pedregosa, F. (2017). Sympy: Symbolic computing in python. *PeerJ Computer Science*, 3, e103. <https://doi.org/10.7717/peerj-cs.103>
- Monod, J. (1949). The growth of bacterial cultures. *Annual Review of Microbiology*, 3(1), 371–394.
- Mundhenk, T., Landajuela, M., Glatt, R., Santiago, C.P., faissol, D., Petersen, B.K. (2021). Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. In: M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan (eds.) *Advances in Neural Information Processing Systems*, vol. 34, pp. 24912–24923. Curran Associates, Inc.
- Petersen, B.K., Larma, M.L., Mundhenk, T.N., Santiago, C.P., Kim, S.K., Kim, J.T. (2019) Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. arXiv preprint [arXiv:1912.04871](https://arxiv.org/abs/1912.04871)

- Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential Evolution: A Practical Approach to Global Optimization*. NY: Springer Science & Business Media.
- Ramm, A. G., & Smirnova, A. B. (2001). On stable numerical differentiation. *Mathematics of Computation*, 70, 1131–1153.
- Romano, J.D., Le, T.T., La Cava, W., Gregg, J.T., Goldberg, D.J., Chakraborty, P., Ray, N.L., Himmelstein, D., Fu, W., Moore, J.H. (2021). Pmlb v1.0: An open source dataset collection for benchmarking machine learning methods. arXiv preprint [arXiv:2012.00058v2](https://arxiv.org/abs/2012.00058v2) (2021)
- Somacal, A., Barrera, Y., Boechi, L., Jonckheere, M., Lefieux, V., Picard, D., & Smucler, E. (2022). Uncovering differential equations from data with hidden variables. *Physical Review E*, 105, 054209.
- Stankovski, T., Duggento, A., McClintock, P. V., & Stefanovska, A. (2014). A tutorial on time-evolving dynamical Bayesian inference. *European Physical Journal: Special Topics*, 223, 2685–2703. <https://doi.org/10.1140/epjst/e2014-02286-7>
- Stolle, R., & Bradley, E. (2007). Communicable knowledge in automated system identification. In S. Džeroski & L. Todorovski (Eds.), *Computational Discovery of Scientific Knowledge: Introduction, Techniques, and Applications in Environmental and Life Sciences* (pp. 17–43). Berlin Heidelberg, Berlin, Heidelberg: Springer.
- Storn, R., & Price, K. (1997). Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341–359.
- Strogatz, S. H. (2018). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Florida: CRC Press.
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... van Mulbregt, P. (2020). SciPy 1.0 contributors: SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- Zheng, P., Askham, T., Brunton, S. L., Kutz, J. N., & Aravkin, A. Y. (2018). A unified framework for sparse relaxed regularized regression: SR3. *IEEE Access*, 7, 1404–1423.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.