Journal of Physics:
Complexity

TOPICAL REVIEW • OPEN ACCESS

Discovering equations from data: symbolic regression in dynamical systems

To cite this article: Beatriz R Brum *et al* 2026 *J. Phys. Complex.* **7** 012001

Journal of Physics: Complexity

TOPICAL REVIEW

Discovering equations from data: symbolic regression in dynamical systems

Beatriz R Brum[1], **Luiza Lober**[1], **Isolde Previdelli**[2] **and Francisco A Rodrigues**[3,*]

[1] Institute of Sciences Mathematics and Computation, Universidade de São Paulo-Campus de São Carlos, Caixa Postal 668, 13560-970 São Carlos, São Paulo, Brazil
[2] Statistic Department, State University of Maringá, Maringá, Brazil
[3] Department of Mathematics Applied and Statistics, Institute of Sciences Mathematics and Computation, Universidade de São Paulo-Campus de São Carlos, Caixa Postal 668, 13560-970 São Carlos, São Paulo, Brazil
* Author to whom any correspondence should be addressed.

E-mail: francisco@icmc.usp.br, beatrizbrum@usp.br and luiza.lober@usp.br

Keywords: symbolic regression, dynamic processes, data driven approaches

OPEN ACCESS

RECEIVED
29 August 2025

REVISED
27 December 2025

ACCEPTED FOR PUBLICATION
28 January 2026

PUBLISHED
13 February 2026

Abstract

The process of discovering equations from data lies at the heart of physics and in many other areas of research, including mathematical ecology and epidemiology. Recently, machine learning methods known as symbolic regression (SR) emerged as a way to automate this task. This study presents an overview of the current literature on SR, while also comparing the efficiency of five state-of-the-art methods in recovering the governing equations from nine processes, including chaotic dynamics and epidemic models. Benchmark results demonstrate the PySR method as the most suitable for inferring equations, with some estimates being indistinguishable from the original analytical forms. These results highlight the potential of SR as a robust tool for inferring and modeling real-world phenomena.

1. Introduction

The discovery of equations from observational data is one of the fundamental pillars of the traditional scientific method. From the work of Johannes Kepler, who inferred the laws of planetary motion from meticulous astronomical observations [1] collected by Tycho Brahe [2], to Isaac Newton's theoretical formulations that consolidated classical mechanics, the process of identifying mathematical relationships underlying natural phenomena has historically been characterized by its systematic trial-and-error procedures.

In the last few decades, the advent of Big Data, characterized by the production and availability of an immense volume of complex, mostly nonlinear data in several fields, has renewed the need for methods capable of uncovering physical laws directly from data. Understanding the intrinsic structure of these datasets and deriving symbolic representations that capture system-level behavior has intensified the demand for advanced analytical techniques tailored to large-scale data.

Modern computational techniques have accelerated the development and use of various techniques to expand regression methods and propose new approaches. In this context, symbolic regression (SR) has emerged as a powerful tool to automate the discovery of mathematical expressions from data. Unlike parametric methods, which are limited to adjusting coefficients in predefined equations and require extensive analysis of statistical significance, goodness-of-fit, and model diagnostics, SR jointly searches for both the structure of the equation and its parameters. This search is performed by exploring a vast combinatorial space composed of mathematical operators, variables, and constants, which gives SR greater flexibility in modeling complex relationships.

SR models can draw from multiple machine learning techniques, including genetic algorithms (GAs) [3], sparse regression [4, 5], neural networks [6] Kolmogorov–Arnold representations [7, 8], and more recently transformer architectures [9]. Some of these are originally conventional algorithms that make no prior assumptions about the model structure [10], limiting human interaction to the definition of basic

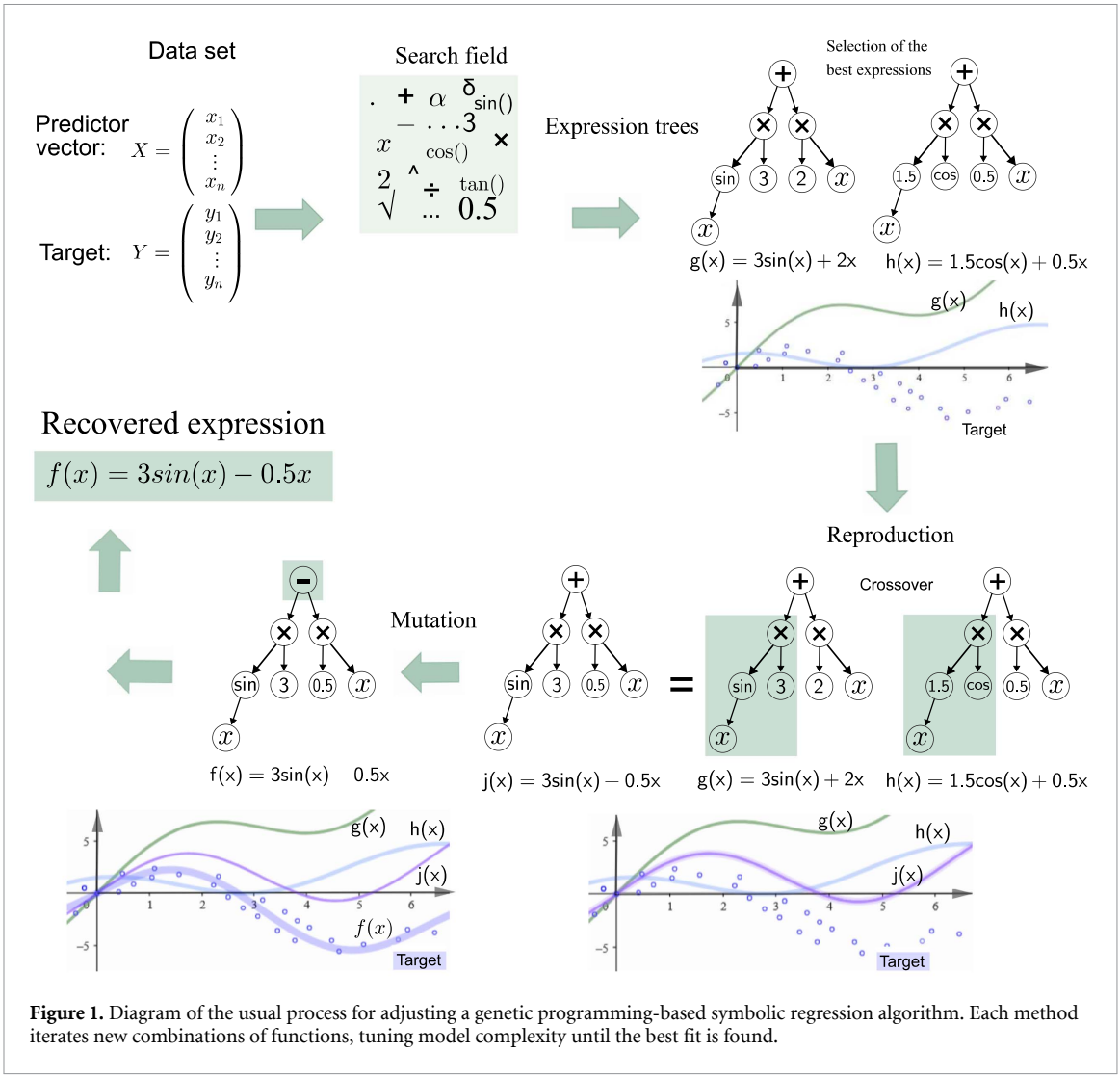

Figure 1. Diagram of the usual process for adjusting a genetic programming-based symbolic regression algorithm. Each method iterates new combinations of functions, tuning model complexity until the best fit is found.

parameters to execute the algorithm. In contrast, newer models allow for more substantive interaction between the user and the machine. These unconventional approaches make it possible to encode prior knowledge about the underlying dynamics, thereby guiding and constraining the search for interpretable symbolic expressions.

Beyond presenting a historical overview on the evolution of SR techniques, this study applied SR to data from different dynamical systems. The objective is twofold: to recover the mathematical equations that govern each system, and to compare the performance of several state-of-the-art SR models. Results show that certain algorithms can reconstruct the structural form of specific systems with high accuracy, and in some cases with expressions indistinguishable from the original equations, suggesting great potential for this set of techniques to be used in real world data in future studies.

1.1. Symbolic regression (SR)

The first supervised learning method capable of inferring governing equations from data was introduced by Langley [11], which was known as the BACON algorithm. However, this framework struggled with equations involving constraints, performed poorly with noisy data, and was limited by the modest computing power available in the early 1970s. These factors ultimately led the research community to set aside the idea of automated equation discovery for several decades.

A few years later, in 1975, John Holland and his colleagues at the University of Michigan introduced the concept of GAs [12]. Inspired by the principles of natural selection [13], GAs provided a robust framework for optimization through evolutionary processes. Building on this idea, Koza [14] introduced genetic programming (GP), which represents candidate solutions as decision trees and iteratively evolves them to discover governing equations for dynamical systems [15]. These advances created the foundation for SR, offering a systematic approach to extract mathematical relationships directly from observational data. The usual pipeline of these algorithms is illustrated in figure 1.

In subsequent years, SR algorithms have gained prominence as they focus on optimization and exploration of approximate solutions for systems with unknown equations of motion [16]. Building on the principles of SR, dissimilarity analysis, and mating strategies, Gustafson *et al* [17] enhanced GP, which yielded demonstrable improvements in accuracy, validated through statistical testing.

A notable milestone in evolutionary SR was introduced by Schmidt and Lipson [18], who proposed a method known as Pareto GP, which is a method based on multi-objective optimization that balances accuracy and complexity. In their study, experimental data was collected from computationally tracked motion, which was then used by the model to find the governing equations of that system through either characteristic Lagrangians or Hamiltonians.

Further progress followed with the development of *Eureqa*, a GP-based algorithm designed to return not only a single expression, but an entire Pareto-optimal set of candidate equations ordered by complexity. Extensive analyses of Eureqa's performance are documented in Dubçáková [19]. Continuing in the same evolutionary tradition, Stephens [3] introduced GPlearn, an open-source Python implementation tightly integrated with `scikit-learn`, facilitating seamless use alongside modern machine learning tools.

Many state-of-the-art SR algorithms continue to build on GP. Among these, PySR stands out as a powerful tool for generating interpretable models [20]. Written in Julia for high-performance computation, PySR is highly optimized while also providing a convenient Python interface, making it both efficient and accessible to researchers.

Parallel to the contributions made by GP, Brunton *et al* [5] combined sparsity-promoting techniques and machine learning to discover the governing equations of dynamic systems from noisy data. This methodology allowed their proposed algorithm to be used both in simple and in high-dimensional systems. For instance, it has several applications in both linear and nonlinear oscillatory dynamics, such as chaotic Lorenz systems and fluid vortex shedding behind an obstacle [5].

Also drawing from sparsity-based algorithms, Mangan *et al* [21] developed an algorithm for implicit ordinary differential equations (ODEs) (implicit-SINDy), a contribution that was essential for the discovery of metabolic and regulatory networks, which often exhibit nonlinear dynamics with rational function nonlinearities in their formulation. This algorithm can be applied to biological networks such as Michaelis–Menten enzyme kinetics, a regulatory in the metabolic network for yeast glycolysis.

In the same group of sparsity-promoting techniques, Rudy *et al* [22] developed a method capable of discovering partial differential equations based on time series measurements, advancing the open field of research of identifiability of dynamical systems.

Exploring the implementation of sparse regularized regression proposed by Zheng *et al* [23], Champion *et al* [24] presented a sparse optimization framework capable of learning parsimonious models of dynamic systems from data, a formulation that aims to discover the equations of a dynamic system through data and by selecting relevant terms from an array of possible functions. This proposal of a comprehensive framework of SR models resulted in the 'sparse identification of non linear dynamics' regression package in Python, also known as PySINDy, which also encompasses various optimizers and necessary tools for the operation of the algorithm. [4, 5, 25, 26].

SR can also benefit from neural network architectures, as first demonstrated by Sahoo *et al* [27]. Their work demonstrated that neural networks can learn functional forms in mechanical systems such as the cart-pendulum, generalizing beyond the observed parameter space. Later, Udrescu and Tegmark [6] introduced AI-Feynman, a multidimensional recursive algorithm combining neural networks with physics-inspired techniques. The approach was validated by effectively discovering all of the 100 selected equations from Feynman's lectures on physics [28], and as a consequence was considered one of the best algorithms available to investigate physical systems, consequently increasing interest of the scientific community in using SR algorithms in this field.

Among approaches based on neural networks, the PyKAN method [7] offers an interesting alternative to the previously mentioned algorithms. Based on the Kolmogorov–Arnold representation theorem, Kolmogorov–Arnold networks (KANs) use learnable univariate functions on the edges of the neural network, combined through multivariate linear operations at the nodes, instead of relying on fixed activation functions as in traditional neural networks. This design provides both high expressivity and interpretability, making them well suited for SR tasks. Extensions such as the MultKAN model [8] represent some of the most recent advances in SR and highlight the rapid evolution of this area.

Another recently proposed avenue for SR leverages transformer-based deep neural networks for equation inference, with methods implemented for both functional [29, 30] and dynamical SR [30, 31]. Unlike the aforementioned approaches featuring neural networks, transformers allow for sequence-to-sequence learning of tasks, that is, they translate the trained weights of the model to a different, unseen task, which reduces computational costs in the inference stage.

1.2. Identifiability of systems

Despite the several approaches to automatic equation discovery that were discussed previously, the question of the overall applicability of SR, or knowing whether a system is uniquely identifiable from data, remains an active research topic, and an essential consideration when applying these methods to unseen, real world data.

For parametric ODEs, as considered in this study, one can answer this question using the framework developed by Qiu *et al* [32], provided that the parameters of those ODEs can assumed linear. Their method also includes quantitative scores that assess the impact of noise in data. The authors also discuss the identifiability for higher-dimensional systems, with the overall conclusions pointing to such systems being identifiable unless their dimensions approach infinity.

In contrast, answering the same question for partial differential equations remains largely unresolved. Scholl *et al* [33] provides a first detailed analysis on this topic, and a few examples of applications are given by Rudy *et al* [22] and Kiyani *et al* [34], but the field is still at an early stage.

More broadly, another relevant issue to equation discovery comes from the need of some systems to apply coordinate transformations to the data to then allow for the equations to be inferred, as was the case for Kepler's laws of planetary motion. Tackling this issue unfortunately requires in-domain knowledge of the system in some cases, although a few methods that will be discussed in the next section do have built-in mechanisms to either simplify equations according to known properties of a system, or use a similar procedure to equation discovery that is usually employed in the domain of physics, which includes transformations to the variables.

2. SR algorithms

The SR models employed in this study are open source algorithms that represent the most recent and significant contributions to the field. These include: (i) GPlearn, an evolutionary algorithm [3], (ii) AI-Feynman, which combines neural networks with algebraic simplification techniques [6, 26], (iii) PySINDy, based on sparse regression [25], (iv) PySR, which leverages GP [20], (v) PyKAN, a neural-network-based approach [7] and (vi) ODEFormer, applying an encoder–decoder transformer algorithm to obtain dynamical systems equations. The following subsections provide a detailed description of each method

2.1. GPLearn

This GP algorithm starts its search by randomly generating a population of symbolic expressions, which are the result of combinations from the search space, or in other words the combination of operators and functions defined by the user, such as $\{+, -, *, \sin, \cdots\}$. Each of those combinations are treated by the algorithm as a single genetic code, represented by a uniquely coded chromosome.

The evaluation of each individual fit, $f(\boldsymbol{x})$, is then performed through transformations, i.e. genotype-phenotype mappings (f_g) and phenotype-fitness (f_p) [35], resulting in the convolution $f = f_p(f_g(\boldsymbol{x}^g))$. The fittest individuals, according to a goodness of fit metric selected by the user, are then selected to keep evolving while the others are eliminated [16]. With a new and smaller population, these individuals undergo a selection process in pairs. In the reproduction stage, the algorithm performs crossovers and mutations to generate a new population. This cycle is repeated until convergence, when an expression that balances accuracy and complexity is obtained.

2.2. AI-Feynman

The AI-Feynman algorithm, developed at MIT's Artificial Intelligence Laboratory, is one of the most widely know SR methods due to its strong descriptive power in physics, as discussed in section 1.1. A key strength of this algorithm lies in the similarity between its procedure and the way physicists traditionally model phenomena. The main advantage it was is the capability of deriving compositional expressions—that is, a function f that can be represented as a combination of a small set of elementary functions, e.g. through linear combinations or other structured compositions.

The set of tools employed by this method ranges from simple processes in the search for symbolic representations, to the application of state-of-the-art machine learning techniques such as neural networks, with the resulting expression being generated through a tree encoding using reverse Polish notation [6]. The algorithm leverages methods that exploit common simplifying properties in natural physical processes, such as dimensional analysis, a straightforward approach resulting in a considerable reduction of the initial variable space, solving the issue of overshooting the complexity necessary for the expressions.

Additionally, it also includes nonlinear least squares polynomial fitting, allowing for the adjustment of polynomials ranging from zero to the fourth degree. The method also uses an exhaustive search algorithm, generating a space that includes all possible expressions, from the simplest to the most complex with varied parameter combinations. Moreover, fitting can also be done using neural network-based structures, also simplifying the investigation of symmetry and separability properties. Finally, the algorithm incorporates equality verification and data transformations, returning a list of expressions ordered by the lowest MDL Loss, a criterion based on information theory that selects the shortest hypothesis that strikes a balance between accuracy and complexity. The final choice between them is up to the researcher.

2.3. PySINDy

This algorithm adds a sparsity constraint in its formulation to identify nonlinear differential equations. This technique also penalizes the error function and facilitates the identification of coefficients with nonlinear behavior by weighting terms according to a set of criteria. The structural form of the aforementioned sparse formulation can be expressed as

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = f(\mathbf{x}(t)). \tag{1}$$

The data describing f is required to be sparse in the state variables $\mathbf{x}(t) \in \mathbb{R}^n$ [36], while the derivative of each of the different variables tends to be sparse in the space of possible functions. For example, in the linear combination below,

$$f_i(x) = \xi_1\theta_1(x) + \xi_2\theta_2(x) + \cdots + \xi_j\theta_j(x), \tag{2}$$

most of the coefficients will end up canceling each other out when an appropriate set of θ_j functions are found.

The method then assumes an approximation of the form

$$\dot{\mathbf{X}} \approx \Theta(\mathbf{X})\,\Xi, \tag{3}$$

where Ξ is a set of coefficients that determines the active terms in f, which must satisfy the sparsity requirement in equation (3) [25, 26].

Thus, the algorithm initializes with the following matrices: $\mathbf{X}$ and $\dot{\mathbf{X}}$. The first one is composed of time series of the variables measured from the system. The second corresponds to the target matrix of time derivatives, generated by differentiating $\mathbf{X}$ through a set of differentiation methods available in the algorithm.

Finally, the matrix $\Theta(\mathbf{X})$ lists the candidate functions for the formulation of the sought symbolic expression, which are user-defined. Through the regularization term $R(\Xi)$, used by the algorithm internally, restrictions are applied to the coefficients of the model's parameters, which characterizes the sparse regression. The parameter λ sets the sparsity threshold, and to achieve this, the software also uses the Pareto curve algorithm, which displays the simplest expression identified through regularization, thereby ensuring a more parsimonious model and reducing overfitting.

2.4. PySR

Based on GAs and implemented in the Julia programming language, PySR's main advantage over alternatives such as GPLearn lies in the use of a framework with higher computational performance offered by this new programming language, while also standing out for its efficiency and robustness, as discussed in [20].

Furthermore, it is possible to implement custom functions in the code using SIMD kernels in Julia. These kernels can be applied at runtime, ensuring high performance by processing multiple operations on data simultaneously, performing automatic differentiation, and handling populations of mathematical expressions, while taking advantage of parallel computing [20]. In the aforementioned paper, a benchmarking tool named 'EmpiricalBench' was also proposed, with the objective of comparing, evaluate and measure the capabilities of different SR algorithms in scientific applications.

Another interesting aspect of the algorithm is that simulations with nested operators can be controlled, ensuring that undesired compositions between functions do not occur, which is a useful tool to reduce expression complexity. The method also allows unary operators to be incorporated based on prior theoretical knowledge about the data, in addition to identifying the simplest expression through a combination of precision and complexity, similar to other algorithms. Furthermore, the method returns a list of the best models found, which can be inspected by the researcher if the automatically selected expression is considered inappropriate when properties of the dynamics studied are known.

2.5. PyKAN

PyKAN was developed by Liu *et al* [7], replacing multilayer perceptron (MLP) neural networks with KANs in its architecture. The motivation is that MLPs are based on the universal approximation theorem, which states that continuous functions can be approximated with arbitrary precision provided that the network has an appropriate topology for that specific dataset. Achieving such a topology, however, is known to be a challenging task, and that often results in highly specialized models that do not generalize well to different problems.

In contrast, the Kolmogorov–Arnold representation theorem guarantees that every continuous function can be expressed as a combination of simpler functions. Given a set of variables derived from a physical process as a continuous function, $f\colon [0,1]^n \to \mathbb{R}$, it can be represented as:

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}\left(x_p\right) \right), \tag{4}$$

where $\Phi_q : [0,1] \to \mathbb{R}$ e $\phi_{q,p} : \mathbb{R} \to \mathbb{R}$ are continuous functions. This theorem provides a theoretical foundation to KANs [7].

In PyKAN's implementation, the authors generalize the original representation, where the depth was 2 layers and the width was equal to $2n+1$ with n representing the number of input variables. By enabling the construction of deeper architectures and training them via backpropagation, PyKAN can capture more complex relationships commonly encountered in real-world applications [7].

KANs also differs from MLPs by not using fixed activation functions. Instead of conventional weights, KANs adopt one-dimensional functions along the edges, which are parameterized by splines during training. These splines are smooth functions that fit the data, allowing KANs to learn their own activation functions and providing greater flexibility in modeling.

KANs also tends to avoid overfitting: when the network is fed with a training dataset, it learns a specific mapping. However, by using another part of the same dataset, the network learns a different mapping. In practice, the network adjusts only a few control points, ensuring that previously learned information is retained. This is made possible through the control points of B-splines, which allow for local adaptation without compromising the knowledge already acquired, an often observed problem with MLPs. During training, the spline grids can be refined, turning them denser and increasing the number of parameters, and as such allowing for more specialized models, allowing the network to specialize while still adjusting primarily a limited number of nodes.

The predominance of the multiplication operation in real physical systems motivated the authors to enhance the algorithm, leading to MultKAN's release. As highlighted by Liu *et al* [8], this new approach has the potential to reveal multiplicative structures present in data. In addition, the authors equipped the algorithm with tools that incorporate prior knowledge about the system under study, such as KANCompiler, providing a significant advantage in discovering symbolic expressions. The method selects the result by minimizing a total cost function that combines the training or test error (MSE) with regularization terms.

2.6. ODEFormer

To infer dynamical systems using deep learning, ODEFormer, proposed by D'Ascoli *et al* [9], is the most up-to-date encoder–decoder transformer method and is supported by an open codebase. Compared with early transformer-based algorithms, ODEFormer has the advantage of being able to infer the governing equations of multidimensional systems, in this way proving to be a particularly well-suited method three dimensional systems and others with several independent variables. Unlike the aforementioned SR algorithms, ODEFormer operates under a different paradigm: its predictive capability is based on dynamics learned during training on large datasets, allowing the algorithm to probabilistically identify the behavior of several systems, later transferred to the dataset of interest.

ODEFormer includes 16 attention heads [37] and 512 embedding dimensions, totalling approximately 86 million parameters. These parameters were trained using an extensive synthetic dataset from various dynamic systems. This data undergo an embedding strategy in which expressions are tokenized and represented as vectors in a space, denoted by $\mathbb{R}^{((D+1)\times 3)\times d_{\text{emb}}}$, where D is the dimension of the original system that generated the data[4], and d_{emb} is the embedding dimension associated with the original equation. The ability to treat data as a sequence of tokens, a fundamental principle of modern machine

[4] $D_{\max} = 6$ for the pre-trained model used in this study.

translation, is what makes transformer-based algorithms unique compared to other SR approaches. To decode and infer equations, the model uses a beam sampling method [38].

The authors also propose an extensive dataset for benchmarking their algorithm, named ODEBench [9]. In it, 63 equations with varying number of dimensions, with some exhibiting chaotic behavior, are used to ascertain the performance of the trained model and released publicly for future benchmarks.

3. Applications

SR has been successfully applied to multiple domains of knowledge over the past decade, providing an overall positive contribution to diverse areas of knowledge. Notable examples include climate system modeling [39], as well the development of hybrid algorithms in materials science [40]. In ecology, Chen *et al* [41] employed GP to understand the dynamics of complex ecosystems, while Abdellaoui and Mehrkanoon [42] used SR modeling in wind speed prediction. In finance, Luo and Yu [43] used SR with the objective of understanding the implied volatility surface in the financial market.

Further applications highlight the versatility of SR methods. Kiyani *et al* [34] applied GPlearn to discover the closed-form of unknown components of complex nonlinear PDEs through a domain decomposition approach. Gudetti *et al* [44] demonstrated that SINDy can be used for NVH (noise, vibration and harshness) applications aimed at vibration control in products. Miyazaki *et al* [45] employed AI-Feynman to discover the hyperbolic discounting formulation that could not be solved analytically previously. In astrophysics, Wong and Cranmer [46] demonstrated that SR through PySR can be used as an effective strategy to identify interpretable gravitational wave population models. Beyond the essence of these applications, there is still other areas that can benefit from this approach, such as epidemic modeling, acoustics and many others.

The effectiveness of SR, however, often depends on the nature and structure of the data, which may favor certain algorithms over others. Although highly impactful works have presented increasingly accurate methods for regression, there are still no single standardized methodology to evaluate their efficiency, due to the diversity and complexity in which each SR model was developed. Some proposed solutions to this problem are discussed below.

Udrescu and Tegmark [6], performed a comprehensive comparison between AI-Feynman and Eureqa [18], using 120 synthetic datasets for this comparative analysis. More recently, La Cava *et al* [47] introduced a reproducible and open-source benchmarking project platform named SRBench, which currently evaluates the performance of 14 contemporary SR methods on 252 datasets, alongside seven machine learning baselines. However, many newer SR algorithms are still missing from this platform at the time of writing.

To better assess the capabilities of newer SR models, particularly in recovering governing equations and in analyzing state transitions in compartmental epidemiological dynamics, a systematic benchmark was conducted. The epidemiological systems selected for this study, together with other representative dynamical systems of interest, are described in the following section.

3.1. Dynamical systems

In this section, four types of dynamic systems will be discussed: two from the domain of physics, one from biology, and the fourth category consisting of another six systems that describe epidemic propagation through different assumptions on its composing compartments.

The systems presented in sections 3.1.1–3.1.3 were chosen for their overarching relevance in the domains of physics and biology, allowing this study to analyze the performance of the SR algorithms in describing a chaotic system, oscillatory motion and predator-prey dynamics, this way anchoring the novel use of SR algorithms to epidemic models being proposed here to the overall performance of those methods in multiple fields.

3.1.1. Lorenz attractor

A classic example of chaotic dynamics that originates from a model to atmospheric convection [48], and independently from single-mode laser dynamics [49], is given by the Lorenz–Haken equations, which are described by

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= -x(\rho-z)-y, \\ \dot{z} &= xy-\beta z \end{aligned} \tag{5}$$

where, considering Lorenz's derivation for the atmosphere, x is proportional to the rate of convection of a fluid, y the horizontal temperature variation and z is the vertical temperature variation. σ, ρ and β are

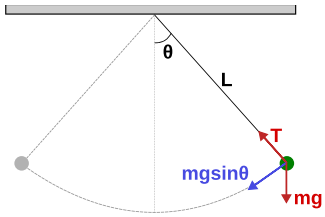

Figure 2. Force diagram of the unitary non-linear pendulum.

constants that relate to properties of that physical system. The system also appears in many other fields and have generalizations to higher dimensions [50].

3.1.2. The non-linear pendulum

Acquiring the equations of motion of this system analytically is relatively simple. To start, drawing the force diagram as done in figure 2 already implies that the sum of forces will be $F = mg\sin(\theta)$. Using Newton's second law of motion and taking $x = r\theta = L\theta$, the previous equation then becomes $F = m\ddot{x} = mL\ddot{\theta} = -mg\sin(\theta)$.

Noting that the angular frequency in this case is $\omega = \sqrt{g/L}$ and $\ddot{\theta} + F(\theta) = \ddot{\theta} + \omega^2\sin\theta = 0$, then by rewriting these terms, one arrives at the following equation:

$$\ddot{\theta} = -\frac{g}{l}\sin\theta, \tag{6}$$

which, together with $u = \dot{\theta}$, can then be used to simulate the behavior of this system.

3.1.3. Lotka–Volterra predator-prey dynamics

An essential baseline model in ecology, the Lotka–Volterra equations are used to simulate and understand the interaction of predator and prey populations in a food chain [51].

Even when making simplistic assumptions on the environment and the populations of the two interacting species, such as ample food being available at all times and that being entirely reliant on the prey population size, no account of genetic variation and adaptation, alongside of no environment changes, the model still is able to capture the general oscillations of population sizes that are observed in nature.

The system of equations that describe this model is given by

$$\begin{aligned} \dot{u} = &\quad \alpha u - \beta uv \\ \dot{v} = &\quad -\gamma v + \delta uv, \end{aligned} \tag{7}$$

where u represents the population density of prey, v the predator's. $\alpha, \beta, \gamma, \delta \in \mathbb{R}^+$, with α as the maximum growth rate of prey per capita, β the effect of predators in the death rate of prey and, complementary to the last two, γ is the predator's per capita death rate and δ the effect of prey in predator's growth rate.

When considering the equilibrium of the two populations, that is, $u = \gamma/\delta$ and $y = \alpha/\beta$ respectively for the prey and predator [52], both depend on each other's parameters, which as a consequence means that increasing the prey growth rate benefits the predator, with that in turn not granting an improvement to the prey population in the long term.

3.1.4. Epidemiological compartmental models

When studying the propagation of diseases in a population, a first step to understand the overall evolution of patients in a group is to compartmentalize each stage of the infection. In doing so, one can not only categorize populations, but also describe rates of transition from each category.

The simplest model that can be built that way considers only two states: susceptible (S) and infected (I) individuals. Assuming the transition rates of infection (β), and recovery (γ), alongside N as the number of individuals, the resulting dynamics will be described by

$$\dot{I} = -\dot{S} = \frac{\beta}{N}IS - \gamma I, \tag{8}$$

where it is assumed that no new individuals enter the group or leave it, or $S(t) + I(t) = N$.

Table 1. All compartmental models used and their governing equations, with transition rates as indicated in figure 3.

SIS	SIR	SIRV	SIRS	SEIR	SEIRD
$\dot{I} = -\dot{S} = \frac{\beta}{N}IS - \gamma I$	$\dot{S} = -\frac{\beta IS}{N}$ $\dot{I} = \frac{\beta IS}{N} - \gamma I$ $\dot{R} = \gamma I$	$\dot{S} = -\frac{\beta SI}{N} - \epsilon S$ $\dot{I} = \frac{\beta SI}{N} - \gamma I$ $\dot{R} = \gamma I$ $\dot{V} = \epsilon S$	$\dot{S} = -\frac{\beta SI}{N} + \delta R$ $\dot{I} = \frac{\beta SI}{N} - \gamma I$ $\dot{R} = \gamma I - \delta R$	$\dot{S} = -\frac{\beta IS}{N}$ $\dot{E} = \frac{\beta IS}{N} - \sigma E$ $\dot{I} = \sigma E - \gamma I$ $\dot{R} = \gamma I$	$\dot{S} = -\frac{\beta SI}{N}$ $\dot{E} = \frac{\beta SI}{N} - \sigma E$ $\dot{I} = \sigma E - (\gamma + \mu)I$ $\dot{R} = \gamma I$ $\dot{D} = \mu I$

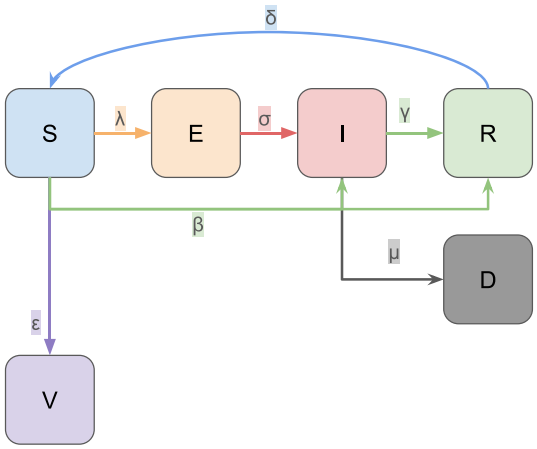

Figure 3. Schematic of the compartmental transition for the selected epidemiological models, which use only parts of all compartments displayed. For their full equations, see table 1.

If the individuals are allowed to recover from that disease, then R will be the total number of those that were cured, resulting in the following set of equations:

$$\begin{aligned} \dot{S} &= -\frac{\beta IS}{N} \\ \dot{I} &= \frac{\beta IS}{N} - \gamma I. \\ \dot{R} &= \gamma I \end{aligned} \tag{9}$$

In both cases, the parameters β and γ are essential to characterize the dynamics, as the basic reproductive number $R_0 = \frac{\beta}{\gamma}$ is reliant on them, and characteristic of the disease's threshold: if each infected individual infects more than one susceptible individual, or $R_0 > 0$, then there's the onset of the epidemic. Otherwise, if $R_0 < 0$, the disease eventually vanishes from the population.

Expanding further from the aforementioned SIR and SIS systems, several other compartmental models can be built using different states for the individuals in a network. In this study, the following ones are used, with their equations shown below in table 1 (see also figure 3).

- SIRV [53], which vaccinated (V) individuals are also accounted for in the dynamics;
- SIRS [54], allowing reinfection in the SIR model;
- SEIR [55], adding an exposed (E) category to the standard SIR;
- SEIRD [56], expanding the SEIR model by adding a deceased (D) count.

Investigating several of these systems allows for broader conclusions on the efficiency of the SR algorithms examined, as these expanded compartmental models aim to account for the various phases and potential developments of an epidemic, drawing closer to real world disease spreading.

4. Methodology

4.1. Generating synthetic data

To compare the six chosen SR methods, data from the dynamic systems described in section 3.1 were generated. All relevant parameters to simulate these systems and to implement each algorithm are listed in appendix.

Table 2. Use of in-domain knowledge of the investigated systems during SR training. A ✓indicates that the method is available to the algorithm, and ✓✓denotes usage for all systems, unless indicated.

Method \ Algorithm	Variable selection	Single equation fitting	Operator selection	Equation fixing
GPLearn	✓✓	✓✓[a]	✓✓	
AI Feynman	✓✓	✓✓	✓✓	
PySINDy	✓		✓✓	✓
PySR	✓	✓✓	✓✓	✓
PyKAN	✓✓	✓✓	✓✓	✓✓
ODEFormer	✓	✓		

[a] Not used for the Lorenz, Lotka–Volterra, Pendulum and SIS systems.

It is important to mention that this study does not aim to provide an exhaustive benchmarking of all possible compartmental models in the literature. Moreover, a thorough benchmark would also require an overall study on the effect of parameter selection to the efficacy of SR techniques, and here performance comparisons were restricted to a given set of constants, including epidemiological transition rates for each case (see appendix for details).

4.2. Use of in-domain knowledge

As discussed previously in section 2, some SR algorithms do have ways to integrate known characteristics of the system under investigation into its regression procedure. When the system is known in its entirety, including its variables and their respective derivatives, the objective of inferring the equation that originated the complete data can be simplified by using in domain knowledge of relationships of such variables.

In this study, five forms of incorporating prior information during training were considered, each implemented differently across models when available. Table 2 below displays the available methods to do so and whether they were used in this study. Those were: (i) whether the variables in the dataset were explicitly selected or split to reflect data that would be descriptive of the known dynamics; (ii) single equation fitting instead of investigating the whole system at once; (iii) selecting the set of operators, which includes mathematical operands and functions; and (iv) choosing the best equation from the list of results of the fitting process, according to what is known of the system, if they were listed as worse according to the model's internal accuracy score. For this last case, only PyKAN required manual adjustments.

It is important to mention that, in most common in real-world scenarios, information about the investigated system will likely be absent. Beyond the inability to fine-tune the algorithms by using known properties, this also introduces a significant challenge: during training, calculating derivatives by finite difference methods can introduce (or amplify) noise in data, which in turn complicates the identification of a system. Schaeffer and McCalla [57] addressed this issue for sparsity-based algorithms through integral formulations that circumvent the need for numerical differentiation, although more studies are needed to further complement other methods.

4.3. Metrics and definition of a baseline

Evaluating the results of a SR method consists in determining whether it is capable of recovering the exact mathematical expression underlying the observed dynamics, that is, verify if the identified expression are be mathematically equivalent to the original equation, along with the estimated parameters approximating the actual values numerically. To this end, this study compared algorithms using three criteria: (i) comparing the structural forms obtained by the SR algorithms with the original ones; (ii) quantifying the difference of the resulting data on the differential equations to the original system's; and (iii) ascertaining the complexity of the equations obtained by the algorithms.

o quantify structural recovery, a Wilcoxon signed-rank test [58] was employed to assess whether the SR-generated expressions produced data statistically indistinguishable from that of the true system. Recovery was considered successful when the null hypothesis of no difference was not rejected at $\alpha = 0.05$ and p-value> 0.05.

To anchor SR performance to other machine learning methods, a random forest [59] algorithm was included as a non-interpretable baseline. For both the original equations and those inferred by SR algorithms, solutions were integrated over the same temporal grid used for data generation, and predictive performance was quantified using the coefficient of determination ($R2$), following standard practice in the SR literature [60].

Lastly, the complexity of each inferred equation was computed following the discussion in [60], where it is defined as the amount of variables, operators and constants present in the given equation. For multidimensional equations, this metric measures the sum of each dimension's complexity.

5. Results and discussions

Following the specifications of section 4, the results in tables 3 and 4 were obtained. For all the tables shown below, dynamic systems whose structural forms were correctly identified, when compared to the original equations, are marked with a 'check mark' (✓), while approximations or incorrect descriptions to these systems are left without these markings.

The results show that all employed SR models, except for ODEFormer and AI Feynman, demonstrated sufficient capability to identify the symbolic representations of the majority of dynamical systems investigated, specially when considering their effectiveness in the description of structural forms from epidemiological dynamics.

PySR was the best performing algorithm in all aspects, successfully identifying the correct structural form of all systems, and all but one with significant differences in the resulting parameters, making this algorithm the most versatile and accurate of all those tested.

PySINDy and PyKAN also identified all systems, but not as accurately when considering the results of the Wilcoxon test, with PyKAN obtaining SIR's system of equations without significant differences. Although these methods produce slightly inferior results according to figure 4, they prove to be sufficiently accurate in describing the system's equations of motion. However, it is worth noting that they were more sensitive to changes in the parameter used during the search: while in KAN stronger regularization favors the recovery of the approximate structural form, in some cases it seems to hinder parameter optimization. In PySINDy, on the other hand, optimizing regularization parameters may be the solution to this problem.

GPLearn only could not recover one set of equations (SEIRD), and although most of its results proved to be significantly different from the original dynamics according to the Wilcoxon test, it is still capable of providing the correct structural form of other systems. However, without proper parameter selection, its output equations tend to be more complex than the original structural forms, featuring excessive multiplication and combination of terms, which can be a result of the way GP methods increase the complexity of equations (see section 2.1).

AI Feynman recovers most equations from the Lorenz, Lotka–Volterra and non-linear pendulum systems, only missing the correct structural form on the z-axis of the former. However, the performance for the compartmental epidemiological systems was subpar, with none of the obtained results reflecting the original dynamics. Inspecting them further, it can be seen that, for SIS, SIR and SIRV, AI Feynman outputs equations of higher complexity than the originating expressions, with the pattern reversed for SIRS, SEIR and SEIRD. This instability in solutions points to the model not being as effective for this particular type of data.

Lastly, ODEFormer only identify one of the systems included in this study, while also having below-average R^2 in most cases. This algorithm also inferred equations with degrees of complexity that in some cases varied significantly when compared to the original equations, with a notable decrease in performance for systems with higher dimensions.

Overall, although some of the resulting equations from the SR algorithms were only approximations to the real ones, the comparison to the originating systems revealed that these differences were generally small, which can be verified when integrating the resulting equations in the same time frame as the original data and calculating R^2 to infer regression efficiency. The results shown in figure 4 indicates that most algorithms can at least effectively capture the overall dynamics and provide an overview at the governing equations for varying phenomena.

5.1. The effects of noise in SR

Comparing the chosen algorithms to a set of dynamical systems also requires understanding the effects of corrupted data to the performance of these methods. To do so, two of such systems, the Lotka–Volterra model for predator-prey dynamics and the SIR compartmental model, were chosen to compare the effects of adding noise to data and the resulting effectiveness of the regression performed by all algorithms. The choice for these two systems was based on the results of table 5, where almost all algorithms effectively obtained the correct equations for both systems. Beyond adjusting the datasets to add gaussian distributed noise, no hyperparameters (see tables 6 and 9) were altered.

Notably, PyKAN was the most affected by noisy data, presenting a complete loss in R^2 for both systems using any amount of noise, pointing to a rather poor generalization of results by this method. In

Table 3. Original structural form of the dynamic systems, alongside relevant parameters used by the dynamic systems to generate their synthetic data (first column), and equations found by each symbolic regression algorithm (all other columns). Dynamical systems whose structural forms were correctly identified are marked with 'check mark' (✓).

Governing equation	GPLearn	AI-Feynman	PySINDy	PySR	PyKAN	ODEFormer
Non linear pendulum	✓	✓	✓	✓	✓	
$\dot{\theta} = \omega$	$\dot{\theta} = \omega$	$\dot{\theta} = \omega$	$\dot{\theta} = \omega$	$\dot{\theta} = \omega$	$\dot{\theta} = \omega$	$\dot{\theta} = 1.06\omega - 0.04\omega(0.05\omega - 13.14\theta)$
$\dot{\omega} = -9.8\sin(\theta)$	$\dot{\omega} = -9.17sin(1.08\theta)$	$\dot{\omega} = -9.8\sin(\theta)$	$\dot{\omega} = -9.8\sin(\theta)$	$\dot{\omega} = -9.8\sin(\theta)$	$\dot{\omega} = -9.8\sin(\theta)$	$\dot{\omega} = -9.41\theta - \frac{1.2027}{(10.73-15.52\theta)}$
Lotka–Volterra	✓	✓	✓	✓	✓	✓
$\dot{u} = 2u - 0.5uv$	$\dot{u} = 2u - 0.5uv$	$\dot{u} = 2u - 0.5uv$	$\dot{u} = 1.94u - 0.49uv$	$\dot{u} = 2.0u - 0.5uv$	$\dot{u} = 2u - 0.5uv$	$\dot{u} = 1.5u - 0.4uv$
$\dot{v} = -v + 0.375uv$	$\dot{v} = -v + 0.18uv$	$\dot{v} = -0.99v + 0.33uv$	$\dot{v} = -0.95v + 0.37uv$	$\dot{v} = 0.37uv - 1.0v$	$\dot{v} = -v + 0.19uv$	$\dot{v} = -1.4v + 0.2uv$
Lorenz			✓	✓	✓	
$\dot{x} = 2(y - x)$	$\dot{x} = 2(y - x)$	$\dot{x} = 2(y - x)$	$\dot{x} = 2(y - x)$	$\dot{x} = 2.0(y - x)$	$\dot{x} = 2(y - x)$	$\dot{x} = 1.9(y - x)$
$\dot{y} = x(1 - z) - y$	$\dot{y} = x(1 - z) - y$	$\dot{y} = x(1 - z)) - y$	$\dot{y} = x(1 - z) - y$	$\dot{y} = x(1 - z) - y$	$\dot{y} = x(1 - z) - y$	$\dot{y} = -1.2yz^2$
$\dot{z} = xy - 2.6z$	$\dot{z} = 0.17y - 0.92z$	$\dot{z} = 0.02 + z\sqrt{z}$	$\dot{z} = xy - 2.6z$	$\dot{z} = xy - 0.6z$	$\dot{z} = -xy - 2.6z$	$\dot{z} = 0.05y - 0.63z$

Table 4. Description of the symbolic representations of the approximated epidemic propagation dynamics. Dynamical systems whose structural forms were correctly identified are marked with 'check mark' (✓).

Governing equation	GPLearn	AI-Feynman	PySINDy	PySR	PyKAN	ODEFormer
SIS	✓		✓	✓	✓	
$\dot{S} = -0.3SI + 0.1I$	$\dot{S} = -0.29SI + 0.1I$	$\dot{S} = -0.3SI + 0.1I^2$	$\dot{S} = -0.3SI - 0.1S$	$\dot{S} = -0.3SI + 0.1I$	$\dot{S} = -0.3SI + 0.1I$	$\dot{S} = 9.01SI(0.03(-1 + 0.14S)^2 - 0.11S)$
$\dot{I} = 0.3SI - 0.1I$	$\dot{I} = 0.29SI - 0.1I$	$\dot{I} = 0.2SI - 0.1I^2$	$\dot{I} = 0.3SI + 0.1S$	$\dot{I} = 0.3SI - 0.1I$	$\dot{I} = 0.3SI - 0.1I$	$\dot{I} = 0.09I(2.1 - 3.44I)$
SIR	✓		✓	✓	✓	
$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.1S$
$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.44SI - 0.1I$	$\dot{I} = SI - I^2 - 0.0007$	$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.5SI - 0.1R$
$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$
SIRV	✓			✓	✓	
$\dot{S} = -0.5SI - 0.2S$	$\dot{S} = -SI - 0.146S$	$\dot{S} = -0.4SI - 0.2S\sqrt{I+1}$	$\dot{S} = -0.5SI - 0.2S$	$\dot{S} = -0.5SI - 0.2S$	$\dot{S} = -0.5SI - 0.2S$	$\dot{S} = -0.3S$
$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.32SI - 0.09I$	$\dot{I} = -0.09I + 0.09S\sqrt{\sqrt{I}+1}$	$\dot{I} = 0.4SI - 0.2IV$	$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.1S - 0.1I$
$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 1.0V - 2.7R$
$\dot{V} = 0.2S$	$\dot{V} = 0.2S$	$\dot{V} = 0.2S$	$\dot{V} = 0.2S$	$\dot{V} = 0.2S$	$\dot{V} = 0.2S$	$\dot{V} = 0.2S$
SIRS	✓			✓	✓	
$\dot{S} = -0.5SI + 0.2R$	$\dot{S} = -0.47SI + 0.19R$	$\dot{S} = -0.02SI(S + I^2 - R) - 0.02S$	$\dot{S} = -0.5SI + 0.2R$	$\dot{S} = -0.5SI + 0.2R$	$\dot{S} = -0.317SI + 0.2R - 0.2$	$\dot{S} = -0.2S^2$
$\dot{I} = 0.5SI - 0.1I$	$\dot{I} = 0.49SI - 0.1I$	$\dot{I} = 1.08I(0.27S - 0.9)$	$\dot{I} = 0.1SI$	$\dot{I} = 0.5SI - I$	$\dot{I} = 0.3SI - 1.0I$	$\dot{I} = 0.3I$
$\dot{R} = 0.1I - 0.2R$	$\dot{R} = 0.10I - 0.21R$	$\dot{R} = I - 0.19R$	$\dot{R} = I + 0.9R$	$\dot{R} = 0.1I - 0.2R$	$\dot{R} = 1.0I - 0.2R$	$\dot{R} = -1.7R$
SEIR	✓		✓	✓	✓	
$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5IS$	$\dot{S} = -1.3SE$
$\dot{E} = 0.5SI - 0.5E$	$\dot{E} = 0.48SI - 0.48E$	$\dot{E} = 0.5SI - 0.5E$	$\dot{E} = 0.5SI - 0.5E$	$\dot{E} = 0.5SI - 0.5E$	$\dot{E} = -0.5E + 0.5IS$	$\dot{E} = 0.1E - 0.7ER$
$\dot{I} = 0.5E - 0.1I$	$\dot{I} = 0.45E - 0.096I$	$\dot{I} = 0.27\left(E + \frac{E^2}{I+E}\right)$	$\dot{I} = 0.5E - 0.1I$	$\dot{I} = 0.5E - 0.1I$	$\dot{I} = 0.5E - 0.1I$	$\dot{I} = 0.4E - 0.1I$
$\dot{R} = 0.1I$	$\dot{R} = 0.096I$	$\dot{R} = \arcsin(0.09I)$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 1.5I - 28.1SIR$
SEIRD			✓	✓	✓	
$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.3SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5SI$	$\dot{S} = -0.5IS$	$\dot{S} = -0.4SI$
$\dot{E} = 0.5SI - 0.2E$	$\dot{E} = 0.1SE - 0.06I$	$\dot{E} = -0.01(E * ((E/I) + 1))$	$\dot{E} = 0.5SI - 0.2E$	$\dot{E} = 0.5SI - 0.2E$	$\dot{E} = -0.2E + 0.5IS$	$\dot{E} = 1.0I - 0.9E$
$\dot{I} = 0.2E - 0.2I$	$\dot{I} = (E + 0.06) * (E - I)$	$\dot{I} = 0.2 * E - 0.2I$	$\dot{I} = 0.2E - 0.2I$	$\dot{I} = 0.2E - 0.2I$	$\dot{I} = 0.2E - 0.2I$	$\dot{I} = 0.1I - 0.8I(-0.2 - 1.0D)^2$
$\dot{R} = 0.1I$	$\dot{R} = 1.0I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1I$	$\dot{R} = 0.1E$
$\dot{D} = 0.1I$	$\dot{D} = 0.1I$	$\dot{D} = 0$	$\dot{D} = 0.1I$	$\dot{D} = 0.1I$	$\dot{D} = 0.1I$	$\dot{D} = 0.1I$

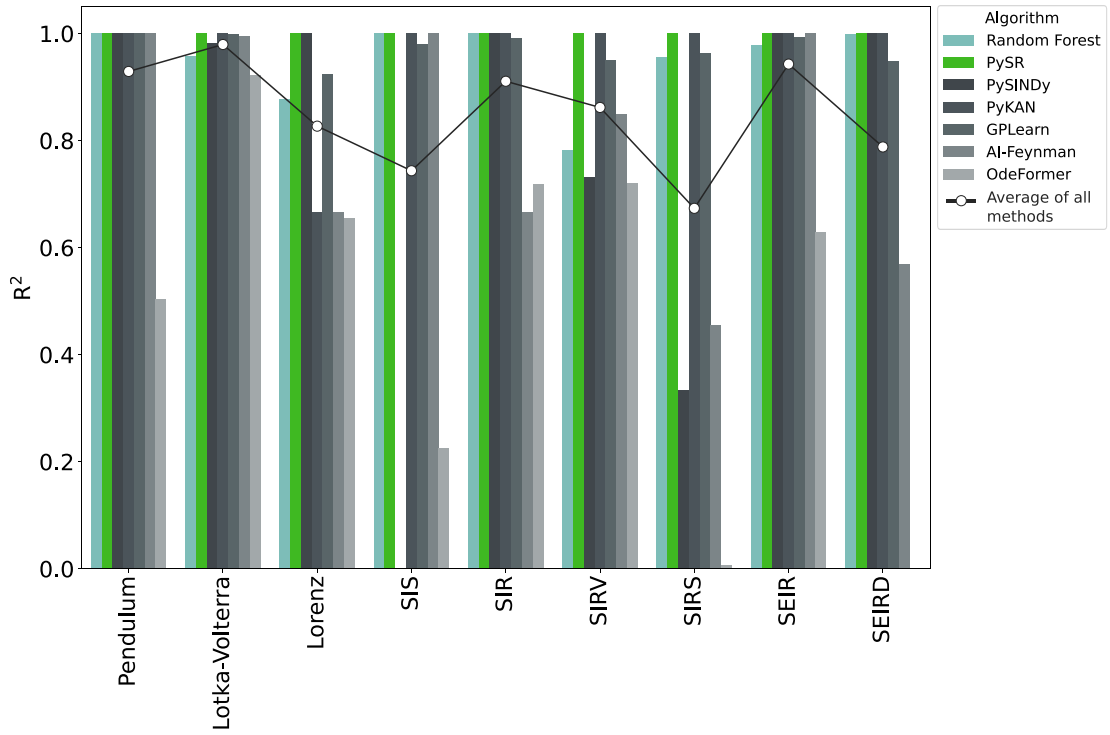

Figure 4. Performance of the symbolic regression models in terms of R^2, with a black line displaying an average over all methods for a given system.

Table 5. Summary of the main results of tables 3 and 4. A ✓ indicates that the structural form of the system was successfully identified, while double ✓s list a result that showed no statistically significant differences compared to the original dynamics, according to the Wilcoxon test. Cells with added numbers show and increase (or decrease) in complexity when compared to the original system.

System		Symbolic regression method					
Name	Complexity	GPLearn	AI-Feynman	PySINDy	PySR	PyKAN	ODEFormer
Non-linear pendulum	8	✓✓	✓✓	✓✓	✓✓	✓✓	+13
Lotka–Volterra	19	✓	✓	✓	✓✓	✓	✓
Lorenz	22	✓✓	−2	✓	✓✓	✓	−3
SIS	20	✓	+4	✓✓	✓✓	✓✓	+3
SIR	21	✓	+2	✓	✓✓	✓✓	+2
SIRV	27	✓	+8	✓✓	✓✓	✓✓	+5
SIRS	28	✓	−2	✓✓	✓	✓✓	+5
SEIR	35	✓	−4	✓✓	✓✓	✓✓	−15
SEIRD	39	−10	−11	✓✓	✓✓	✓✓	−6

contrast, PySR was the least affected by noise in data, with almost negligible differences in R^2 for all cases.

AI-Feynman, PySINDy and GPLearn also presented spikes of performance loss, although not as severe as in PyKAN's case and mostly contained in the Lotka–Volterra system.

ODEFormer had below-average R^2 loss, with an exception for SIR with 1% added noise: in it, the model had $R^2 = 0$ for two components of that system, which presented a significant increase in complexity (37 against 21) compared to the baseline SIR system.

It is important to mention that the results of figure 5 are not equivalent to a systematic benchmark of these techniques: the aim of this particular study was to only increase irregularities on the original data and verify differences in performance. To aim for a thorough benchmark, it would be essential to consider the extensions and settings that can be used to attenuate the effect of noise during their execution, which are present in most SR methods investigated. Below, algorithms that include such strategies to tackle noisy data are discussed:

- The original authors of ODEFormer [9] do thorough comparisons by adding different amounts of noise to several systems, showing that the algorithm is robust to noise up to $\sigma = 0.05$ (or 5% added noise) over several systems;

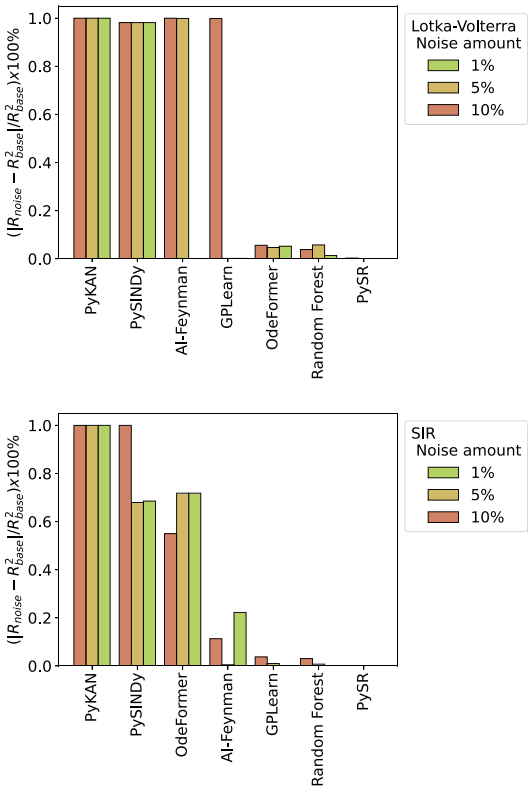

Figure 5. Resulting differences in R^2 when adding Gaussian noise to the synthetic data of a (top) Lotka–Volterra and (bottom) SIR system.

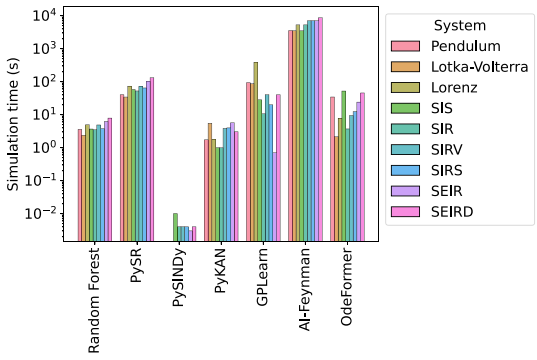

Figure 6. Processing time for each combination of algorithm and dynamical system, in seconds. For PySINDy, $t = 0$ s when simulating a Lorenz, Lotka–Volterra and simple pendulum systems.

- PySINDy provide noise mitigation strategies on their differentiation methods [61], which is the most sensitive to noise stage of this algorithm;
- Raghav *et al* [62] recently extended the original GPLearn method to perform more accurately on noise-sensitive tasks, while also enhancing the original strategy to be able to accommodate user interactivity;
- PySR has a built-in denoise module that can be applied by setting the flag `denoise = True` when initializing the model. More details on the implementation of this method is given on [20].

5.2. Computational complexity

All algorithms are implemented in different ways, and as such the amount of resources, including operation time, may vary. To quantitatively ascertain their computational complexity according to these needs, each individual simulation was timed, with results shown in figure 6.

AI-Feynman is, by far, the slowest of the methods analyzed, with each system requiring up to two hours and a half to be executed. The second slowest method is GPLearn, which averaged 75 s for each simulation. Contrasting these numbers, PySINDy ran almost instantly for some systems, and kept a

very low time average of 5×10^{-3} seconds for others. The rest of the algorithms ran from around a few seconds up to two minutes. These figures are proportionally similar to those listed in the benchmark by La Cava *et al* [60], although the dataset sizes and computer specifications[5] from both studies are different.

6. Conclusions and outlook

This study provided a review of both the historic background of SR and state-of-the-art algorithms currently used by this emerging technique, while also benchmarking several of them in the task of recovering the governing equations of non-linear dynamics. Among all available SR algorithms, GPLearn, AI-Feynman, PySINDy, PySR, PyKAN and ODEFormer were selected for these detailed comparisons, using synthetic data of physical, biological and epidemiological systems, with the latter being an application still underexplored for these models. The methodology employed, along with the results obtained through it, allowed the identification of the best overall SR algorithms, while also presenting an efficient application of such techniques in a new domain.

PySR proved to be the most robust algorithm, recovering the correct structural form in all tested systems and also featuring as the best performing across all metrics: its ability to consistently reconstruct the underlying structure from the data, combined with its computational efficiency, consolidated its superior performance. Only in one case the resulting expressions show statistically significant differences from the original dynamics, although these discrepancies were numerically insignificant.

PySINDy and PyKAN, algorithms that allow for greater interactivity on their training process, also performed well, correctly identifying most dynamic systems. However, in a slightly larger number of cases, the results generated by these methods exhibited significant differences compared to the original equations. It must also be noted that both, and specially PyKAN, had a significant drop in performance when corruption in the form of noise was added to the original data, pointing to a lesser capability for generalization when compared to the top performing method.

Other methods had varying degrees of success, with use cases that can surpass the top performing approaches or supplement them. However, ODEFormer and AI Feynman were two notable outliers when compared to all others. The former, being the first multivariate transformer-based methodology available, thus allowing for inference flexibility that can be further improved upon, has as its current role limited to that of a hypothesis generator, which corroborates [9]. Moreover, although this transfer learning method avoids noise amplification problems from differentiation methods (see section 4.2, it may also represent a comparative disadvantage in comparison to algorithms that incorporate a slice of the system's directly in their training stages [9].

As for AI Feynman, it should be noted that it is the most resource intensive considering the amount of time required to perform simulations. They are also the oldest implemented algorithms from this selection and, in AI-Feynman's case, it is important to mention that the source code, made available by the authors of the methodology themselves in [63], has not been maintained in a few years, with the last update dating from 2020. Future users of the code must be aware that technical difficulties may arise from the algorithm relying on dependencies that are out of date.

The results of this study highlights the potential of SR, suggesting its potential as a new methodology for the modeling of data related to epidemic dynamics, and likely being flexible enough for other domains of knowledge, as current literature indicates. Moreover, updates to these algorithms or the development of new ones could bring even better results when comparing to the already top performing methods, overcoming current limitations and enabling more accurate dynamic recovery and better forecasting results in the future. The continuation of this study is essential to assess the capacity of these methods in describing dynamic systems and, consequently, their contribution to the description of epidemic spreading in real populations.

As for the current limitations of such approach, it is important to mention that even with the current advances to this machine learning field, recovering equations from data through SR cannot be done in polynomial time, characterizing it as a NP-hard problem [64]. This should be taken into consideration when adding large sets of basis functions and their several permutations to these methods.

Moreover, this study did not exhaustively benchmarked all methods included, and given the data driven nature of the approach, expanding the comparisons to a larger array of parameters and initial conditions of the chosen systems, and also adding others, would be necessary to further discuss whether

[5] Computer specifications: Fedora Linux 42 (64-bit), 24 × AMD Ryzen 9 9900X 12-Core Processor, 64GB RAM @3600MHz, NVIDIA GeForce RTX 2060 SUPER, B850M D3HP Motherboard.

a certain method is superior to another, and in which cases. This is specially true when dealing with systems with known chaotic behavior, which includes the investigated Lorenz attractor. Future work can expand on this topic, possibly as an interactive benchmark platform analogue, or as a complement to, SRBench [60].

Finally, although the current results point to a largely successful use of these methods for several systems, the performance for real world datasets can suffer from both limited knowledge of the system's true behavior and the possible imprecision of numerical differentiation methods. Multicollinearity between variables must also be assessed through the appropriate statistical methods, such as variance inflation factors and others [65, 66], before applying these algorithms to unknown datasets. In conclusion, future research employing this method for equation discovery should consider the most robust and up to date techniques to tackle these issues, alongside a thorough investigation of the identifiability of the system in question before providing concluding evidence as to the usefulness of SR to real world data.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/luizalober/review_symb_regression [67].

Acknowledgments

Beatriz Brum thanks CAPES for the financial support provided (Process Number 33001014). Luiza Lober thanks São Paulo Research Foundation (FAPESP) through Grants 2022/16065-3 and 2013/07375-0. Francisco A Rodrigues acknowledges CNPq (Grants 308162/2023-4 and 408389/2024-9) and FAPESP (Grants 20/09835-1 and 13/07375-0) for the financial support given for this research. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES) - Finance Code 001.

Author contributions

Beatriz R Brum
Data curation (equal), Investigation (equal), Methodology (equal), Validation (equal), Visualization (equal), Writing – original draft (equal), Writing – review & editing (equal)

Isolde Previdelli
Conceptualization (equal), Investigation (equal), Supervision (equal), Writing – original draft (equal), Writing – review & editing (equal)

Appendix. Parameter tables

Tables 6 describe the parameters, initial conditions and time intervals used to generate synthetic data. The SOLVE_IVP function from the `SciPy` library was used to numerically integrate these equations over time. The selection of these parameters was made in a case-by-case basis, with the intent of representing a realistic scenario for each system.

As for the SR algorithm's settings, tables 8 and 9 list the employed parameters for the non-linear pendulum, Lotka–Volterra and Lorenz systems, and the epidemiological models, in that order. Any other available parameters that were not mentioned were kept as default. These settings were chosen to allow for the best performance possible, and kept the same when investigating the impact of noise in these algorithms (see section 5.1).

Table 6. Parameters employed for solving the Lorenz attractor, non-linear pendulum and predator-prey (Lotka–Volterra) dynamics.

Parameters	Nonlinear pendulum	Lotka–Volterra	Lorenz
Initial conditions	$\omega = 0$ $\theta = 45$	$u = 20$ $v = 5$	$x_0 = 0.6$ $y_0 = 2.0$ $z_0 = 1.0$
Coefficients	$g = 9.8$ $l = 1.0$	$\alpha = 2.0$ $\beta = 0.5$ $\gamma = 1.0$ $\delta = 0.375$	$\sigma = 2.0$ $\rho = 1.0$ $\beta = 2.6$
Simulation window	[0,5]	[0,7.5]	[0,5]
Time step size	2×10^{-3}	1×10^{-1}	2×10^{-3}

Table 7. Parameters employed to generate data on the chosen compartmental epidemic models (3.1.4). Compartment sizes are listed as fractions of the total number of individuals.

Parameters	SIS	SIR	SIRV	SIRS	SEIR	SEIRD
Initial compartment size	$S_0 = 0.99$ $I_0 = 0.01$	$S_0 = 0.99$ $I_0 = 0.01$ $R_0 = 0$	$S_0 = 0.94$ $I_0 = 0.01$ $R_0 = 0$ $V_0 = 0.05$	$S_0 = 0.99$ $I_0 = 0.01$ $R_0 = 0$	$S_0 = 0.8$ $E_0 = 0.1$ $I_0 = 0.1$ $R_0 = 0$	$S_0 = 0.99$ $E_0 = 0$ $I_0 = 0.01$ $R_0 = 0$ $D_0 = 0$
Coefficients	$\beta = 0.3$ $\gamma = 0.1$	$\beta = 0.5$ $\gamma = 0.1$	$\beta = 0.5$ $\gamma = 0.1$ $\epsilon = 0.5$	$\beta = 0.5$ $\gamma = 0.1$ $\delta = 0.2$	$\beta = 0.5$ $\sigma = 0.5$ $\gamma = 0.1$	$\beta = 0.5$ $\sigma = 0.2$ $\gamma = 0.1$ $\mu = 0.1$
Simulation window	[0, 100]	[0,75]	[0, 35]	[0,60]	[0,80]	[0120]
Time step size	1×10^{-1}	1×10^{-1}	1×10^{-1}	1×10^{-1}	1×10^{-1}	1×10^{-1}

Table 8. Settings and parameters used when running the selected SR algorithms for the Lorenz attractor, non-linear pendulum and predator-prey (Lotka–Volterra) dynamics.

SR model	Parameters	Dynamic System		
		Lorenz	Non-linear pendulum	Lotka–Volterra
GPLearn	population_size	5000	5000	5000
	generations	50	100	50
	tournament_size	50	100	50
	stopping_criteria	0.01	0.01	0.01
	p_crossover	0.7	0.7	0.6
	p_subtree_mutation	0.2	0.2	0.2
	p_hoist_mutation	0.01	0.01	0.01
	p_point_mutation	0.09	0.09	0.09
	init_depth	(2, 6)	(2, 2)	(8, 9)
	parsimony_coefficient	0.001	0.009	0.001
	function_sett	—	$[\times, +, -, \div, \sin]$	$[\times, +, -]$
	random_state	0	0	0
PySINDy	library_functions	All but $f(x) = x^n$	Fourier (n_frequencies = 1) Polynomial (degree = 1)	Polynomial (degree = 3)
	feature_library	x, y, z	θ, ω	u, v
	differentiation_method		FiniteDifference	FiniteDifference (order = 2)
	Optimizer *optmizer*	*STLSQ*	*SR3*	*SR3*
	Optimizer *threshold*	0.2	0.4	0.6
	Optimizer *alpha*	1×10^{-4}	—	1×10^{-4}
	Optimizer *normalize_columns*	*False*	—	*True*
	Optimizer *thresholder*	—	$'l1'$	—
	Optimizer ν	—	—	1
	Optimizer *tol*	—	—	1×10^{-6}
PySR	Population	[30, 30, 30]	[1, 30]	
	N interation	[30,30,30]	[30,30]	[1050 800]
	Binary-Operator	[[-, +, *],[-, +, *],[-, +, *]]	[[*, +],[+, *]]	[[*, +],[-, *]]
	Unary-Operator	[ ],[ ],[ ]	[ ],["sin"]]	[[ ],[ ]]
	Nested_constraints	no	no	no
PyKAN	Network topology	x : *kanpiler* y : *kanpiler* z : *kanpiler*	$\theta : [1, 1]$ $\omega : [1, 1]$	U : *kanpiler* V : *kanpiler*
	seed	0	12	0
	λ	1×10^{-25}	1×10^{-3}	1×10^{-35}
	steps	50	30	60
	λ_coef	1×10^{-19}	1×10^{-15}	1×10^{-2}
ODEFormer	Beam size	100	100	100
	Temperature	0.1	0.1	0.1

Table 9. Settings and parameters employed by the SR algorithms when running the selected SR algorithms for the epidemiological systems.

		Epidemiological models					
SR model	Parameters	SIR	SIS	SEIR	SEIRD	SIRV	SIRS
GPLearn	population_size	6.5×10^2	6.5×10^2	1×10^3	[no, 5×10^2, 1×10^3, 1×10^3, 1×10^3]	[2×10^3, 1×10^3, 1×10^3, 1×10^3]	[5×10^2,8×10^3,1×10^4]
	generations	150	120	[1×10^2,2×10^2,1×10^2,1×10^2]	[1×10^2, 1×10^2, 1×10^2, 2×10^2, 1×10^2]	[3.5×10^2, 5×10^2, 5×10^1,5×10^1]	[1×10^2,1×10^2,6×10^2]
	tournament_size	150	120	[1×10^2,2×10^2,1×10^2,1×10^2]	[no, 1×10^2, 1×10^2, 2×10^2, 1×10^2]	[3.5×10^2, 1.5×10^2, 5×10^1,5×10^1]	[5×10^1,1×10^2,5×10^2]
	stopping_criteria	[1×10^{-2}	[1×10^{-2}	[1×10^{-3}, 1×10^{-3}, 1×10^{-3}, 1×10^{-2}]	[no, 1×10^2, 1×10^2, 2×10^2, 1×10^2]	[1×10^{-2}, 1×10^{-2}, 1×10^{-2}, 1×10^{-3}]	[no,1×10^3,1×10^3]
	p_crossover	0.7	0.6	0.7	0.7	0.7	0.7
	p_subtree_mutation	0.1	0.2	0.1	0.1	0.1	0.1
	p_hoist_mutation	0.1	0.01	[0.1, 0.1, 0.05, 0.05]	[0.05,0.05,0.05,0.1,0.1]	0.1	0.1
	p_point_mutation	0.1	0.09	0.1	0.1	0.1	0.1
	init_depth	no	no	[(2,4),(3, 5),(2,4),(0,5)]	[(no,(3,4),(2,4),(1,2),(1,2)]	[(2,4),(3,3),(2,4),(2,4)]	[no,(3,3),(3,4)]
	parsimony_coefficient	1×10^{-2}	1×10^{-4}	[1×10^{-6}, 1×10^{-5}, 1×10^{-3}, 1×10^{-2}]	[no, 1×10^{-4}, 1×10^{-4}, 1×10^{-1}, 1×10^{-1}]	0.01	[no, 1×10^{-1}3, 1×10^{-1}6]
	function_sett	[×, +]	[×, +, −, ÷]	[[×, −], no,[×, −], [×, −]]	[[×, −], [×, −], [×, −], [×, −], [×, −]]	[[×, −], [×, −], [×, −], [×, −]]	[[×, +], [×, −], [×, −, ¬]]
	random_state	0	0	0	0	0	0
AI-Feynman	BF_try_time	[60s, 60s, 60s]	60s	[240, 300, 3600, 60]	[3600, 360, 60, 60, 60]	[300 300 360 060]	[360 060 600]
	BF_ops_file_type	7ops	7ops	7ops	7ops	7ops	7ops
	polyfit_deg	2	2	2	[2,2,2,2,1]	2	2
	NN_epochs	[400 400 4000]	400	[600 4000 600 600]	[1000 600 4000 600, 600]	[4000 4000 600 1000]	[600 600 600]

(Continued.)

Table 9. (Continued.)

PySINDy	Functions	$x+y, x*y$	$x+y, x*y$	$x, x*y$	$x, x*y$	$x, x*y$	$x, x*y$
	Optimizer	STLSQ	Orthogonal Matching Pursuit	STLSQ	STLSQ	STLSQ	STLSQ
	Threshold	0.1	0.6	0.15	0.053	0.08	0.04
	α	1×10^{-2}	—	1×10^{-3}	1×10^{-3}	1×10^{-5}	1×10^{-4}
	n_nonzero_coefs	—	2	—	–	—	–
	normalize_columns	True	—	True	True	True	False
	Differentiation method	Finite difference, order 2					
PySR	Maxsize	10	10	10	10	10	10
	ninteration	1×10^{3}	1×10^{3}	1×10^{3}	1×10^{3}	1×10^{3}	1×10^{3}
	Binary-Operator	[+, *, -]	[+, *, -]	[+, *, -]	[+, *, -]	[+, *, -]	[+, *, -]
	Unary-Operator	[[ ],[ ],[ ]]	[[ ],[ ]]	[[ ],[ ],[ ],[ ]]	[[ ],[ ],[ ],[ ]]	[[ ],[ ],[ ],[ ]]	
PyKAN	Network topology	[kanpiler, kanpiler, [1,1]]	[kanpiler, kanpiler]	[kanpiler, kanpiler, kanpiler, [1,1]]	[kanpiler, kanpiler, kanpiler, [1,1], [1,1]]	[kanpiler, kanpiler, [1,1], [1,1]]	[kanpiler, kanpiler, [2,1]]
	seed	0	12		12	12	12
	λ	$[1 \times 10^{-15}, 1 \times 10^{-15}, 1 \times 10^{-3}0]$	$[1 \times 10^{-4}5, 1 \times 10^{-25}]$	$[1 \times 10^{-10}, 1 \times 10^{-10}, 1 \times 10^{-10}, 1 \times 10^{-2}0]$	$[1 \times 10^{-10}, 1 \times 10^{-10}, 1 \times 10^{-7}, 1 \times 10^{-2}0, 1 \times 10^{-2}0]$	$[1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-2}0, 1 \times 10^{-2}0]$	$[1 \times 10^{-5}, 1 \times 10^{-3}, 1 \times 10^{-15}]$
	steps	[50, 30 300]	[60,60]	[40 4040 300]	[40,40 50 300 300]	[50 020 300 300]	[500 50 050]
	λ_coef	$[1 \times 10^{-5}, 1 \times 10^{-5}, 1 \times 10^{-10}]$	$[1 \times 10^{-2}, 1 \times 10^{-1}2]$	$[1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-10}]$	$[1 \times 10^{-3}, 1 \times 10^{-3}, 1 \times 10^{-15}, 1 \times 10^{-10}, 1 \times 10^{-10}]$	$[1 \times 10^{-15}, 1 \times 10^{-15}, 1 \times 10^{-10}, 1 \times 10^{-10}]$	$[1 \times 10^{-1}, 1 \times 10^{-15}, 1 \times 10^{-10}]$
ODEFormer	Beam size	100	100	100	100	100	100
	Beam temperature	0.1	0.9	0.1	0.1	0.1	0.1

References

[1] Camps-Valls G, Gerhardus A, Ninad U, Varando G, Martius G, Balaguer-Ballester E, Vinuesa R, Diaz E, Zanna L and Runge J 2023 arXiv:2305.13341
[2] Gleiser M 2005 *The Dancing Universe: From Creation Myths to the Big Bang* (UPNE)
[3] Stephens T 2016 Genetic programming in Python, with a scikit-learn inspired API: GPLearn (available at: https://gplearn.readthedocs.io/en/latest/)
[4] Brunton S L, Proctor J L and Kutz J N 2016 *IFAC-PapersOnLine* **49** 710
[5] Brunton S L, Proctor J L and Kutz J N 2016 *Proc. Natl Acad. Sci.* **113** 3932
[6] Udrescu S-M and Tegmark M 2020 *Sci. Adv.* **6** eaay2631
[7] Liu Z, Wang Y, Vaidya S, Ruehle F, Halverson J, Soljačić M, Hou T Y and Tegmark M 2024 arXiv:2404.19756
[8] Liu Z, Ma P, Wang Y, Matusik W and Tegmark M 2024 arXiv:2408.10205
[9] D'Ascoli S, Becker S, Mathis A, Schwaller P and Kilbertus N 2023 arXiv:2310.05573
[10] Minnebo W and Stijven S 2011 Empowering knowledge computing with variable selection-On variable importance and variable selection in regression random forests and symbolic regression *PhD Thesis, Master's Thesis* University of Antwerp
[11] Langley P 1977 *Int. Joint Conf. on Artificial Intelligence* (Citeseer) p 344
[12] Golberg D E 1989 *Addion Wesley* **1989** 36
[13] Darwin C 2005 On natural selection (Penguin)
[14] Koza J R 1992 *Stat. Comput.* **4** 87–112
[15] Diveev A and Shmalko E 2021 *Machine Learning Control by Symbolic Regression* (Springer)
[16] Sivanandam S and Deepa S 2008 *Introduction to Genetic Algorithms* (Springer) pp 15–37
[17] Gustafson S, Burke E K and Krasnogor N 2005 *2005 IEEE Congress on Evolutionary Computation* (IEEE) pp 02–05
[18] Schmidt M and Lipson H 2009 *Science* **324** 81
[19] Dubčáková R 2011 Eureqa: software review *Genet. Program. Evolvable Mach.* **12** 173–8
[20] Cranmer M 2023 arXiv:2305.01582
[21] Mangan N M, Brunton S L, Proctor J L and Kutz J N 2016 *IEEE Trans. Mol. Biol. Multi-Scale Commun.* **2** 52
[22] Rudy S H, Brunton S L, Proctor J L and Kutz J N 2017 *Sci. Adv.* **3** e1602614
[23] Zheng P, Askham T, Brunton S L, Kutz J N and Aravkin A Y 2018 *IEEE Access* **7** 1404
[24] Champion K, Zheng P, Aravkin A Y, Brunton S L and Kutz J N 2020 *IEEE Access* **8** 169259
[25] De Silva B M, Champion K, Quade M, Loiseau J C, Kutz J N and Brunton S L 2020 arXiv:2004.08424
[26] Kaptanoglu A A *et al* 2021 arXiv:2111.08481
[27] Sahoo S, Lampert C and Martius G 2018 *Int. Conf. on Machine Learning* (PMLR) pp 4442–50
[28] 2024 The Feynman Lectures on Physics (available at: www.feynmanlectures.caltech.edu) (Accessed 7 November 2024)
[29] Valipour M, You B, Panju M and Ghodsi A 2021 arXiv:arXiv.2106.14131
[30] Kamienny P-A, D'Ascoli S, Lample G and Charton F 2022 *Guide Proc.* (Curran Associates Inc.) pp 10269–81
[31] Vastl M, Kulhánek J, Kubalík J, Derner E and Babuška R 2022 arXiv:2205.15764
[32] Qiu X, Xu T, Soltanalizadeh B and Wu H 2022 *Appl. Math. Comput.* **430** 127260
[33] Scholl P, Bacho A, Boche H and Kutyniok G 2023 *ICASSP 2023 - 2023 IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Proc.* (Institute of Electrical and Electronics Engineers Inc.)
[34] Kiyani E, Shukla K, Karniadakis G E and Karttunen M 2023 arXiv:2305.10706
[35] Rothlauf F 2006 *Representations for Genetic and Evolutionary Algorithms* (Springer) pp 9–32
[36] Quade M, Abel M, Kutz J N and Brunton S L 2018 *Chaos* **28** 063116
[37] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L and Polosukhin I 2017 arXiv:1706.03762
[38] Van Gael J, Saatci Y, Teh Y W and Ghahramani Z 2008 *ACM Other Conf.* (Association for Computing Machinery) pp 1088–95
[39] Stanislawska K, Krawiec K and Kundzewicz Z W 2012 *Comput. Math. Appl.* **64** 3717
[40] Wang Y, Wagner N and Rondinelli J M 2019 *MRS Commun.* **9** 793
[41] Chen Y, Angulo M T and Liu Y-Y 2019 *BioEssays* **41** 1900069
[42] Abdellaoui I A and Mehrkanoon S 2021 *2021 IEEE Symp. Series on Computational Intelligence (SSCI)* (IEEE) pp 01–08
[43] Luo J and Yu C L 2023 *Mathematics* **11** 2108
[44] Gudetti J P, Yazdi S J M, Baqersad J, Peters D and Ghamari M 2023 Data-driven modeling of linear and nonlinear dynamic systems for noise and vibration applications *Technical Report* (SAE Technical paper)
[45] Miyazaki M, Ishikawa K-I, Nakashima K, Shimizu H, Takahashi T and Takahashi N 2023 *Front. Artif. Intell.* **6** 1039438
[46] Wong K W and Cranmer M 2022 arXiv:2207.12409
[47] La Cava W, Orzechowski P, Burlacu B, de França F O, Virgolin M, Jin Y, Kommenda M and Moore J H 2021 arXiv:2107.14351
[48] Lorenz E N 1963 *J. Atmos. Sci.* **20** 130
[49] Haken H and Sauermann H 1963 *Z. Phys.* **173** 261
[50] Shen B-W 2025 *16th Chaotic Modeling and Simulation Int. Conf.* (Springer) pp 589–610
[51] Diz-Pita E and Otero-Espinar M V 2021 *Mathematics* **9** 1783
[52] Wangersky P J 1978 *Ann. Rev. Ecol. Syst.* **9** 189
[53] Oke M, Ogunmiloro O, Akinwumi C and Raji R 2019 *Commun. Math. Appl.* **10** 717
[54] Hu H, Yuan X, Huang L and Huang C 2019 *Math. Biosci. Eng.* **16** 5729
[55] Annas S, Pratama M I, Rifandi M, Sanusi W and Side S 2020 *Chaos Solitons Fractals* **139** 110072
[56] Korolev I 2021 *J. Econ.* **220** 63
[57] Schaeffer H and McCalla S G 2017 *Phys. Rev. E* **96** 023302
[58] Conover W J 1999 *Practical Nonparametric Statistics* 3rd edn (Wiley) p 350
[59] Breiman L 2001 *Mach. Learn.* **45** 5
[60] La Cava W, Orzechowski P, Burlacu B, de Franca F, Virgolin M, Jin Y, Kommenda M and Moore J 2021 *Proc. Neural Information Processing Systems Track on Datasets and Benchmarks* vol 1
[61] Kaptanoglu A A *et al* 2022 *J. Open Source Softw.* **7** 3994

[62] Raghav S S, Kumar S T, Balaji R, Sanjay M and Shunmuga C 2024 *ACM Conf.* (Association for Computing Machinery) pp 2076–82
[63] Udrescu S M and Tegmark M 2020 AI-Feynman (available at: https://github.com/SJ001/AI-Feynman) (Accessed 22 July 2025)
[64] Virgolin M and Pissis S P 2022 *Trans. Mach. Learn. Res.* (available at: https://openreview.net/forum?id=LTiaPxqe2e)
[65] Chan J Y-L, Leow S M H, Bea K T, Cheng W K, Phoong S W, Hong Z-W and Chen Y-L 2022 *Mathematics* **10** 1283
[66] Castillo F A and Villa C M 2005 *ACM Conf.* (Association for Computing Machinery) pp 2207–8
[67] Brum B and Lober L 2025 GitHub repository for all code used in this study (available at: https://github.com/luizalober/review_symb_regression/)