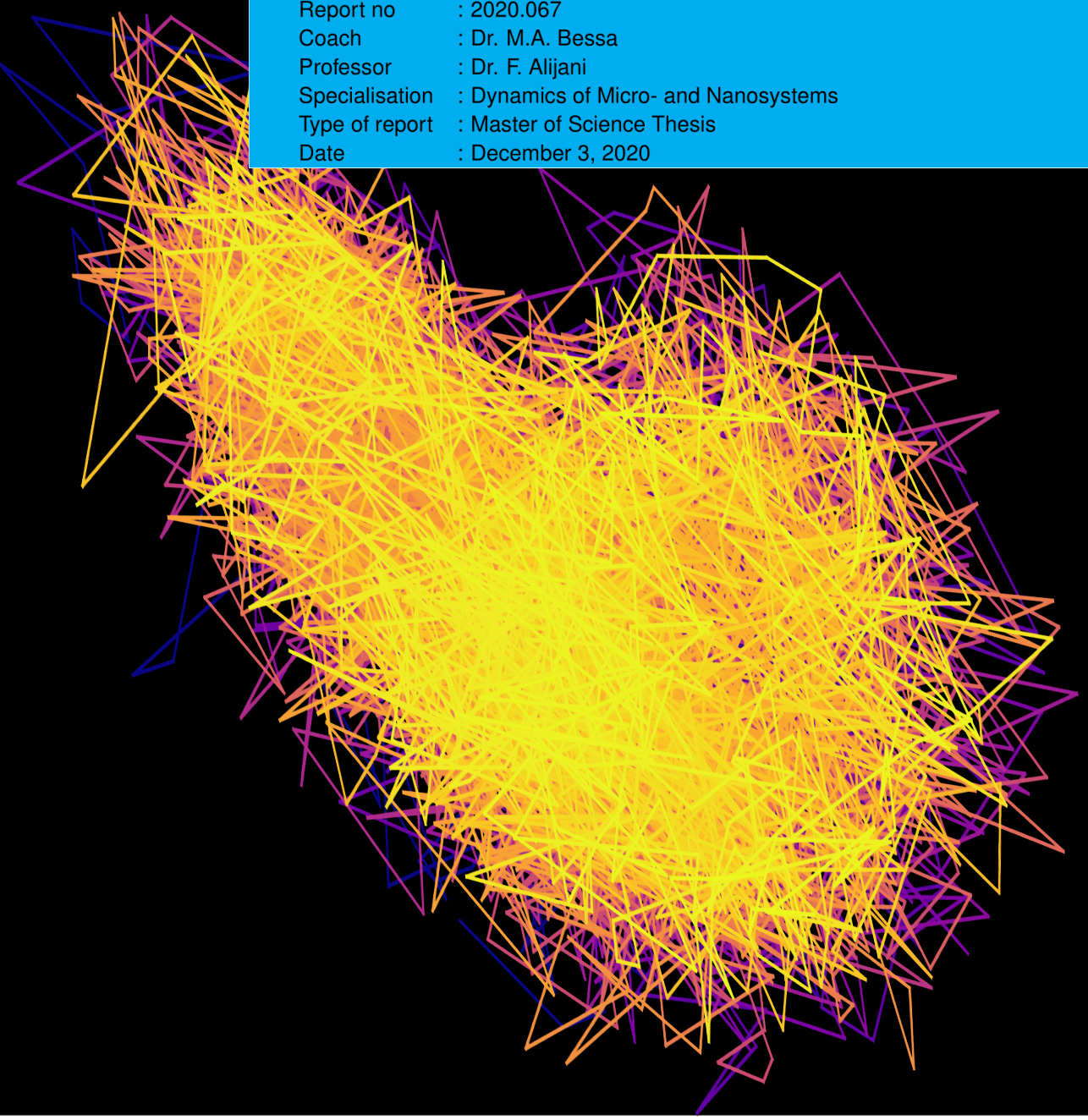
Department of Precision and Microsystems Engineering

Data-driven techniques for finding governing equations of noisy nonlinear dynamical systems

H.C. Lingmont

Report no : 2020.067
Coach : Dr. M.A. Bessa
Professor : Dr. F. Alijani
Specialisation : Dynamics of Micro- and Nanosystems
Type of report : Master of Science Thesis
Date : December 3, 2020



TU Delft Delft University of Technology

Challenge the future

Data-driven techniques for finding governing equations of noisy nonlinear dynamical systems

Department of Precision and Microsystems Engineering

by

Hidde Lingmont

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday December 16, 2020 at 09:30 AM.

Student number: 4390016

Thesis committee:

Dr. F. Alijani,	TU Delft, supervisor
Dr. M.A. Bessa,	TU Delft, supervisor
Dr. A.M. Aragón,	TU Delft, external examiner

TU Delft

Preface

Before you lies my master thesis report. It was written in a period when there was a global pandemic, which caused me to do most of my work from home. It would have been substantially more difficult if it wasn't for the support of my supervisors, Farbod and Miguel. In the online meetings, they have been incredibly helpful, critical where needed and enthusiastic. It was their enthusiasm that motivated me and gave me the strength to continue in the solitude of my room.

I was part of the Dynamics of Micro and Nanosystems (DMN) group at the department of Precision and Microsystems Engineering (PME) and the Bessa group at the department of Materials Science and Engineering at the Mechanical, Maritime and Materials Engineering (3ME) faculty. It was inspiring to be part of these groups and to be part of progress.

Lastly, I would like to thank my parents, my brother, my girlfriend and my friends. Their mental support has helped in many different ways. Without them, I wouldn't be who I am today and I wouldn't be where I am today.

H.C. Lingmont
Delft, December 2020

Summary

With governing equations, we can predict how the dynamics of a system evolve over time. Studies have shown that it is possible to reverse this process and obtain governing equations from data provided by the system. These studies have demonstrated promising results on synthetic data with symbolic regression. However, in experimental settings, the results are often disappointing because noise corrupts the data. Further complications are that symbolic regression often discovers multiple possible equations and that numerically calculating the derivative amplifies the noise. Therefore, the aim of this project is threefold. Firstly, the aim is to make symbolic regression more robust to noise. Secondly, the aim is to determine a single equation from a list of possible equations found with symbolic regression. Thirdly, the aim is to numerically calculate derivatives of data in a way that minimises the amplification of noise.

It was observed that in the field of machine learning, there exist statistical approaches to regression that can accurately capture a system's dynamics, even in the presence of noise. It was investigated how these methods can be used, together with symbolic regression, to increase the noise robustness. We then studied how these machine learning models can be used to numerically calculate derivates. It was further investigated how optimisation processes can be used to choose optimal hyperparameters of Symbolic regression methods and thus single out an equation. In order to explore the bounds, we investigated multiple types of equations, different types and amounts of noise, different regression methods, different symbolic regressors and different optimisation methods.

It was found that Gaussian processes are able to predict a smoothened version of noisy data that often are closer to the noise-free signal than the noisy signal. This smoothened predicted data provides a good basis for numerical differentiation. Since Gaussian processes provide smooth predictions, there is little amplification of error caused by sudden changes in data due to noise. It was also found that equations can be distinguished from each other through their coefficient of determination (R^2) if the predictions are extrapolated far away enough from the training data. The hyperparameters can then be optimised to find the best scoring equation.

We showed that, with these methods, the governing equations can be discovered with significantly higher amounts of noise. The strategy's full potential is demonstrated with the combination of Gaussian process regression and SINDy but the strategy has also been shown to be succesful with different combinations of noise-reduction tools and symbolic regressors. The increased robustness to noise makes symbolic regression more feasible in experimental settings.

Contents

I

Introduction

1

Introduction

The central challenge in finding equations or natural laws from data is widely applicable in many fields of science and engineering. In the age of computers, a paradigm shift emerges in finding equations from physical phenomena with data-driven methods. Advances in machine learning allow for accurately capturing a system's dynamics, even in the presence of noise. However, these methods provide little to no insight in the physics of the system and often do not extrapolate well. Recently methods have been proposed to find equations from data. Once the equation of a system is found, the equation can be used to gain further understanding of the system and the equation can be used to extrapolate. These symbolic regressors show good potential for noiseless synthetic data but are often lacking in experimental settings. In this work, advances in machine learning techniques and symbolic regression are combined to create a white box model that is robust to noise.

A method that successfully discovers governing equations from data could accelerate research in many fields because these algorithms usually run in the order of hours or even minutes. In contrast, scientists can take years to discover governing equations through traditional research. A computer can go through larger amounts of data and discover structures that would be impossible to see without these methods. This could generate new insights in new and existing phenomena.

This thesis is divided in three parts. Chapter 2 and 3 form part I. In chapter 2 a review of the current methods of regression and symbolic regression is presented. Additionally, a knowledge gap is identified. Relevant modelling techniques are discussed. In chapter 3 the research question, research objectives and research methodology are presented. In part II, a paper is presented that investigates how data-driven techniques can be used to improve symbolic regression. More specifically, it is investigated how Gaussian processes aid in reducing the effect of noise on symbolic regression, how the derivatives of a dynamical system can be robustly

and numerically calculated and how machine learning tools can be used to get consistent results in symbolic regression. In the last part, part III, the results are discussed and conclusions are drawn. Furthermore, ideas for further research are discussed in section 5.2.

2

Literature review

For millennia scientists have observed phenomena and tried to understand why certain events and processes happen in a given way. Traditionally the equations to describe these events and processes were found through a thorough understanding of the physics and rigorous research. However, a paradigm shift is emerging from recent advances in machine learning and computing power that enable the use of large datasets to find equations governing a system from observations. With the combination of curiosity, imagination and recognising patterns humans have driven innovation throughout the ages. Nevertheless, there are countless examples of computers finding new patterns that have led to new human understanding.

In the field of system identification, lots of advances have been made to estimate parameters and even the form of the equations of a system. An example is the NARMAX model, which was proposed in [1] and [2]. The NARMAX model can represent a wide range of nonlinear functions by creating a general form:

$$\begin{aligned} y(k) =& F\left[y(k-1), y(k-2), \ldots, y\left(k-n_y\right), u(k-d), u(k-d-1), \ldots, u\left(k-d-n_u\right)\right. \\ & \left. e(k-1), e(k-2), \ldots, e\left(k-n_e\right)\right] + e(k) \end{aligned} \tag{2.1}$$

F() is a nonlinear function, where $y(k)$, $u(k)$ and $e(k)$ represent the output, input and noise respectively. The parameters n_y, n_u and n_e represent the maximum lag and d represents a delay. Often these methods require some knowledge of the system beforehand to make the right assumptions and they do not always give more insight into the physics governing the systems. NARMAX is often used for control in (partly) unknown systems.

Along with NARMAX, there exist many system identification techniques in the frequency domain [3]. The frequency domain can even be useful for denoising data, for example, by applying a Butterworth low-pass filter and filtering the noise at

higher frequencies. Since more literature exists on time series regression, this paper will focuss on time domain methods for a general algorithm. However, it should be kept in mind that some specific problems might be more easily identified in the frequency domain.

Many time series regression methods are black-box identification techniques, which provide no symbolic formulas but do provide excellent fittings. An example is the use of splines. Splines divide the data into bins and find either a linear or polynomial fitting per bin. It then imposes boundary conditions to ensure smoothness. Natural splines are linear outside of the defined region. This means natural splines do not perform well outside of the defined region for non-linear functions.

Another example of black-box models for regression is neural networks. Neural networks are shown to be able to uniformly approximate any continuous function of n real variables in the unit hypercube [4]. They are therefore good universal approximators. Many forms of neural networks exist, such as feedforward, recurrent, convolutional, wavelet, Bayesian and fuzzy neural networks. The mentioned forms of neural networks can be used for regression.

The distribution of a fully-connected neural network with a single hidden layer, which has an i.i.d. prior over parameters, converges to a Gaussian Process in the limit of infinite layer width [5]. Gaussian process regression provides an excellent fitting to a data set, given that the data set is small (<10,000 samples). A problem with larger data sets is that Gaussian process prediction typically scales as $\mathcal{O}(n^3)$. For large data sets storing the Gram matrix and solving the associated linear systems are prohibitive on modern workstations [6].

Recently, advances have been made in symbolic regression [7–9]. These methods provide us with a way of finding the equations governing a system or a natural law and therefore provide us with more insight into the physics. There are, however, still some major problems with these methods. They do not always converge to the right form, are too computationally expensive or do not work in an experimental setting. Noise impairs the ability of these algorithms to find accurate results.

In this section an overview is given of regression and symbolic regression methods, a knowledge gap and current problems are identified, and a proposed way to improve the status quo is given. To answer if the current methods of finding governing equations from experimental data of a system can be improved, this section will look at whether we can improve current methods of symbolic regression, look at other system identification techniques to compare to the status quo and to see if elements of these techniques can be incorporated to the symbolic regression techniques.

2.1. Noisy data

Noise plays an important role in figuring out the right equations as it can cause the methods to find the wrong solution. This happens because various equations can be found that seem to fit the data well. However, many equations are actually overfitting the data and do not perform well outside the given dataset. Noisy measurements are especially prone to overfitting. Overfitting occurs when too many terms are included and the fitting has low bias and high variance. Conversely, a model can also be underfitted, meaning the trend of the data is not captured correctly. High-dimensional data requires many more samples to prevent underfitting.

In figure 2.1 it is shown how data can be fitted in three ways. The figure shows how achieving a higher accuracy with too many terms can result in overfitting and too little terms can result in underfitting.

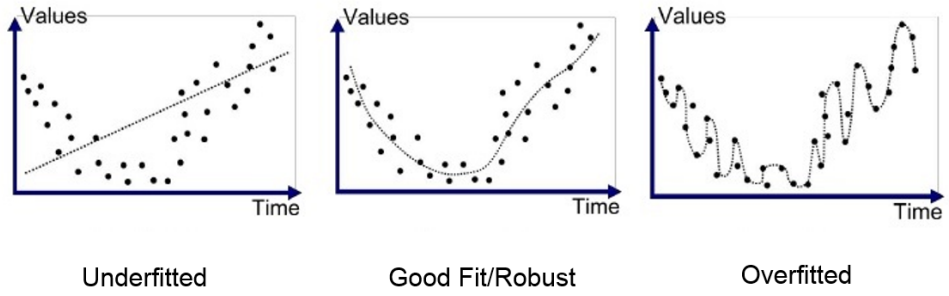


Figure 2.1: An example of overfitting, Reprinted from medium.com, by A. Bhande, 2018, https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

2.2. Regression models and noise reduction

A large number of regression methods provide a good fit to the data but are not well suited for estimating the form of the governing equation. However, these regression models are still relevant as they can be used as a preprocessor. The reason for this is twofold. Firstly, it is hypothesised that symbolic regressors become less prone to overfitting with noisy data if the data is smoothened through one of these regression models. Secondly, as explained in appendix A.2, for higher-order differential equations one still needs access to the derivatives. If these cannot be obtained through experiments, these need to be numerically calculated. Since numerical differentiation can be very prone to noise it also benefits from a smoothening preprocessor. It is therefore proposed to first preprocess the data with one of these regression models. This combination of smoothening and symbolic regression is identified as a research gap.

2.2.1. Common regression models

There exist several regression models. A few common ones are listed below:

- Ordinary least-squares regression
- LASSO regression
- Ridge regression
- Elastic-net regression
- Linear regression
- Polynomial regression
- Splines

Ordinary least-squares forms the basis for many of these models. LASSO regression, ridge regression and elastic-net regression add a penalty term to the least-squares problem. In lasso regression or L1 regularization, the objective is to solve equation 2.2 [10]. This penalises the sum of all absolute values of the weights. In ridge regression or L2 regularization, the penalty term is the sum of the squares of the weights times the hyperparameter λ. Elastic-net regression is a combination of both. Without the penalty terms least-squares regression often overfits a problem by adding large and numerous weights. Lasso promotes more parsimonious models than ridge regression as more weights are driven to 0 and thus creates more sparsity. LASSO regression is therefore used more for problems where it is unclear if all parameters are important.

$$\min_{\beta_0,\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \beta_0 - x_i^T \beta \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq t \tag{2.2}$$

Linear regression and polynomial regression fit an equation to the data. Splines combine linear and polynomial regression and apply it to bins. Splines can also be rational. In splines, smoothness is often ensured by setting boundary conditions between the bins. Natural splines are linear outside of the defined region. These three methods are parametric. Linear and polynomial regression already assume the data follows a specific equation and splines do not generalise well outside the defined region. Non-parametric methods such as neural networks do not assume a specific form of the equation.

2.2.2. Neural networks

Neural networks can be trained to accurately describe the dynamics of a system given only the positional data [11]. Even when features for machine learning are automatically synthesised, neural networks outperform expertly designed traditional statistical models in terms of prediction [12]. The most common form of a neural network is a feedforward neural network. This is often used for simple regression problems. Besides feedforward neural networks there are other forms for regression problems, such as:

Name	Definition
Gaussian	$\kappa(r) = \mathrm{e}^{-(\varepsilon r)^2}$
Multiquadric	$\kappa(r) = \sqrt{1 + (\varepsilon r)^2}$
Inverse Multiquadric	$\kappa(r) = \frac{1}{\sqrt{1+(\varepsilon r)^2}}$
C^0 Matérn	$\kappa(r) = \mathrm{e}^{-\varepsilon r}$
C^2 Matérn	$\kappa(r) = \mathrm{e}^{-\varepsilon r} \cdot (1 + \varepsilon r)$
C^4 Matérn	$\kappa(r) = \mathrm{e}^{-\varepsilon r} \cdot \left(3 + 3\varepsilon r + (\varepsilon r)^2\right)$

Table 2.1: Some common radial basis functions [15].

- Convolutional neural networks
- Wavelet neural networks
- Bayesian neural networks
- Fuzzy neural networks
- Residual neural networks

Bayesian neural networks attempt to bring neural networks to a statistical framework, which yields more parsimonious models and allows to quantify model uncertainty. This is done by adding a distribution over the weights and/or the output. Overfitting is avoided through appropriate priors, which strongly penalise engineered features from deeper layers. This can be done because the architecture of the network is not prespecified but instead optimises the number of layers, the number of features within a layer and the activation functions [13].

Radial basis function networks (RBFN) are common black-box function operators. Training is faster in radial basis function networks than multi-layer perceptrons (MLP). Radial basis function networks are neural networks that use radial basis functions (RBF) as activation functions. A common radial basis function is the Gaussian:

$$\kappa\left(v_i\right) = \exp\left(-\frac{\left\|v_i - c_i\right\|^2}{2\sigma^2}\right) \tag{2.3}$$

However, there exist many more radial basis functions, as shown in table 2.1, or the more recently proposed computationally efficient radial basis function (SQ-RBF), which speeds up training time by about 8% compared to a gaussian radial basis function [14]:

$$\kappa(x) = \begin{cases} 1 - \frac{x^2}{2} & : |x| \leq 1 \\ \frac{(2-|x|)^2}{2} & : 1 \leq |x| \leq 2 \\ 0 & : |x| \geq 2 \end{cases} \tag{2.4}$$

Wavelet neural networks (WNN) are similar to RBFN but use different activation functions. An example of a feedforward WNN is shown in figure 2.2. There are many variants for the mother wavelet Ψ but equation 2.5, known as the Mexican Hat Function, is shown to work satisfactorily in various applications [16]

$$\psi\left(z_{ij}\right) = \left(1 - z_{ij}^2\right) e^{-\frac{1}{2} z_{ij}^2} \tag{2.5}$$

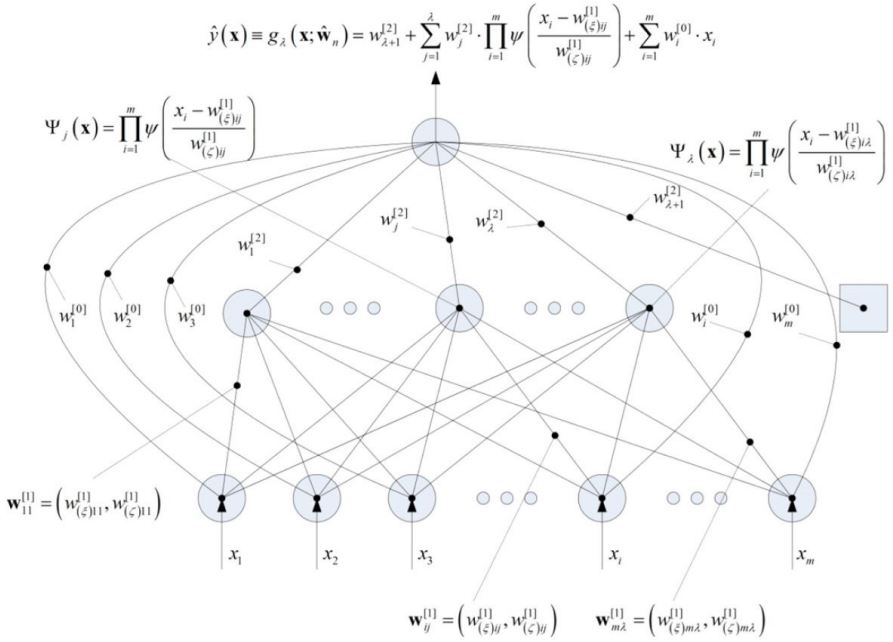


Figure 2.2: A feedforward wavelet neural network, Reprinted from "Wavelet neural networks: A practical guide", by A. K. Alexandridis et al., 2013, Elsevier, Volume 42, Pages 1-27, Copyright 2013 Elsevier Ltd.

WNN performs better in approximating periodic functions and the sharp spike in a piecewise function, whereas RBFN has a higher accuracy for exponential functions [17].

2.2.3. Gaussian Processes

Unlike many regression models, Gaussian process regression does not attempt to learn every parameter in a function. Instead it creates a distribution over a function where it takes a Bayesian approach that is non-parametric. Many samples are taken from a high-dimensional multivariate Gaussian distribution and then it finds the correlation between the samples through kernel functions that determine a measure of closeness. Subsequently, the average of the many samples is taken, which acts

as the prediction line, with a confidence interval. An example of the result is shown in Figure B.1. On the left, it can be seen that the confidence interval is 0 on training points with no noise and quite large where there are no training points. As can be seen on the right, increasing the number of training points result in a far more accurate fitting. A Gaussian process is defined as follows:

$$f(\mathrm{x}) \sim \mathcal{GP}(m(\mathrm{x}), K(\mathrm{x}, \mathrm{x}')) \tag{2.6}$$

An example of a kernel function is shown in equation B.2, where each element in kernel matrix K is defined as shown.

$$k(x, x_j) = \sigma_f^2 e^{\frac{1}{-2l^2}(x_i - x_j)^2} \tag{2.7}$$

With Gaussian processes we are interested in the mean and the covariance, which, for the given data y(zero-mean), are calculated at M unseen inputs as shown in equations B.3 and B.4 [18].

$$\mu_* = \mathrm{K}_{MN} \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{y} \tag{2.8}$$

$$\Sigma. = \mathrm{K}_M - \mathrm{K}_{A/N} \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{K}_{NM} \tag{2.9}$$

In equation B.2, σ_f and l are hyperparameters, which change the predictive quality and uncertainty. The best values for these hyperparameters, which are grouped in θ, are usually found through an optimisation process of the log marginal likelihood [6]

$$\log Z(\theta) = -\frac{1}{2}\left[\mathrm{y}^\top \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{y} + N \log(2\pi) \; + \log\left|\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right|\right] \tag{2.10}$$

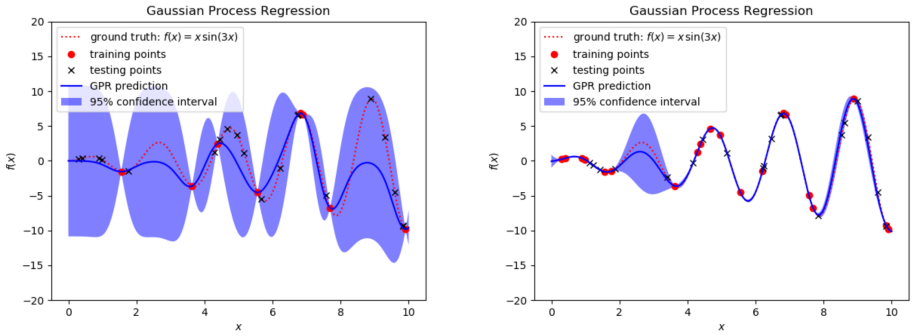


Figure 2.3: An example of Gaussian Processes, Reprinted from imechanica.org, by M.A. Bessa, 2020, https://imechanica.org/node/23957

2.2.4. Choice of smoothener

Gaussian processes are the preferred smoothener, as they are well understood, intuitive and adaptive. Furthermore, its stochastic nature allows for uncertainty quantification. Compared to the described common regression models, it provides better accuracy. In comparison to neural networks, it is faster, more intuitive, less of a black-box and often more accurate. However, it works best with less than 10,000 samples because Gaussian process prediction typically scales as $\mathcal{O}(n^3)$. For larger data sets storing the Gram matrix and solving the associated linear systems are prohibitive on modern workstations [6]. This has been addressed with sparse Gaussian processes [19–21]. A weighted updating scheme for sparse online Gaussian Processes could be used to enable a trade-off during optimization between overall predictive accuracy and a specific focus on better-performing regions of the parameter space [22]. This is useful to address the issue that traditional sparse Gaussian processes perform poorly if the sparse set selection does not preserve resolution in promising areas, such as local optima, to preserve accuracy elsewhere [22].

2.3. Symbolic regression

Three ways of tackling symbolic regression are: SINDy [7, 23, 24], genetic programming [8, 9] and AI Feynman [25]. These methods try to discover a symbolic relationship from the provided data.

2.3.1. SINDy

The SINDy (Sparse Identification of Nonlinear Dynamics) algorithm was proposed in ref. [7]. Here it is described how governing equations of the form $\frac{d\mathrm{x}(t)}{dt} = \mathrm{f}(\mathrm{x}(t))$ can be found from data. In the scripts from ref [7], the following structure is followed:

1. Generate data
2. Compute derivative
3. Build a library of nonlinear time series
4. Sparse regression: sequential least-squares
5. Integrate using identified system
6. Visualise results

Here the library of nonlinear time series can be described as follows:

$$\Theta(\mathrm{X}) = \begin{bmatrix} | & | & | & | & & | & | & \\ 1 & \mathrm{X} & \mathrm{X}^{P_2} & \mathrm{X}^{P_3} & \cdots & \sin(\mathrm{X}) & \cos(\mathrm{X}) & \cdots \\ | & | & & | & & | & | & \end{bmatrix} \tag{2.11}$$

Where $\mathrm{X}^{\mathrm{P_n}}$ denotes polynomial nonlinearities in the state x. For example, $\mathrm{X}^{\mathrm{P_2}}$ is:

$$\mathrm{X}^{P_2} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \cdots & x_2^2(t_1) & \cdots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \cdots & x_2^2(t_2) & \cdots & x_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & \cdots & x_2^2(t_m) & \cdots & x_n^2(t_m) \end{bmatrix} \tag{2.12}$$

The equation that SINDy discovers is then represented in the form:

$$\dot{\mathrm{X}} = \Theta(\mathrm{X})\Xi + \eta Z \tag{2.13}$$

Where ηZ represents the noise and where Ξ determines what terms are active in Θ.

$$\Xi = \Theta(\mathrm{X})^{-1}\dot{\mathrm{X}} \tag{2.14}$$

The sparse regression is done through sequential least-squares. Here, the values for Ξ are found. If we rewrite equation 2.13 to 2.14 and ignore the noise, we can use it as an initial guess for Ξ. If only a constant and linear terms are added to the library, this is the same as linear regression.

Due to the noise many small terms will remain. In order to obtain a more parsimonious model, a sparsification knob is introduced. This sparsification knob is a threshold that will set all values of Ξ below the threshold to zero. Equation 2.14 will then be repeated with the bigger terms until convergence is reached. As an example, the schematic of the SINDy algorithm is shown on the Lorenz equations in figure 2.4.

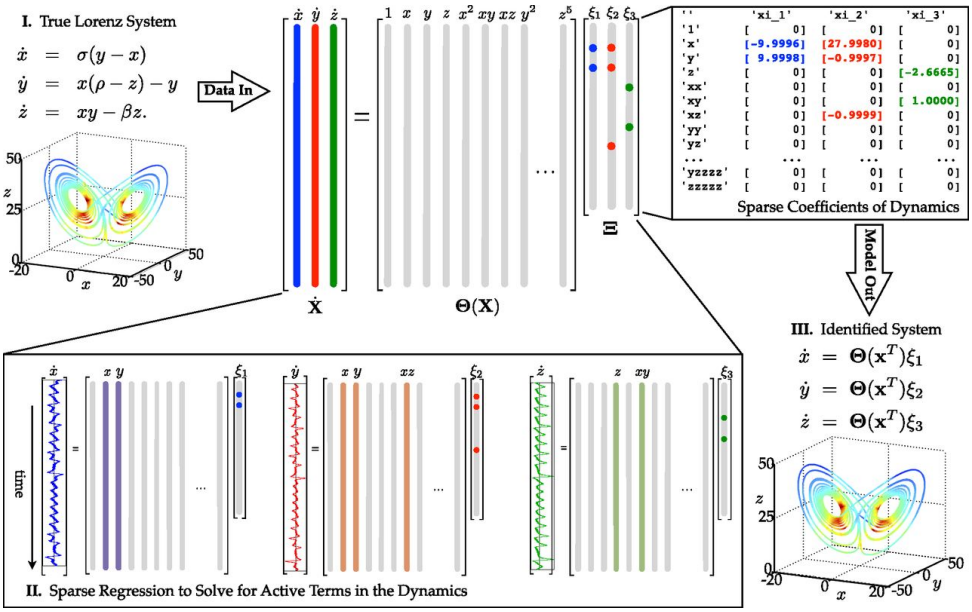


Figure 2.4: Schematic of the SINDy algorithm on the Lorenz equations, Reprinted from "Discovering governing equations from data by sparse identification of nonlinear dynamical systems", by S. L. Brunton et al., 2016, PNAS, Volume 113, Pages 3932–3937

The library of nonlinear time series is predefined. In the original library, only polynomials and trigonometric polynomials were considered. This library can be extended with more combinations but this greatly increases its size. The algorithm is also less accurate if more combinations are added. For example, if we apply the SINDy algorithm to a single pendulum, with a small amount of added noise, it can find the correct form of the equation and the coefficients with a slight error.

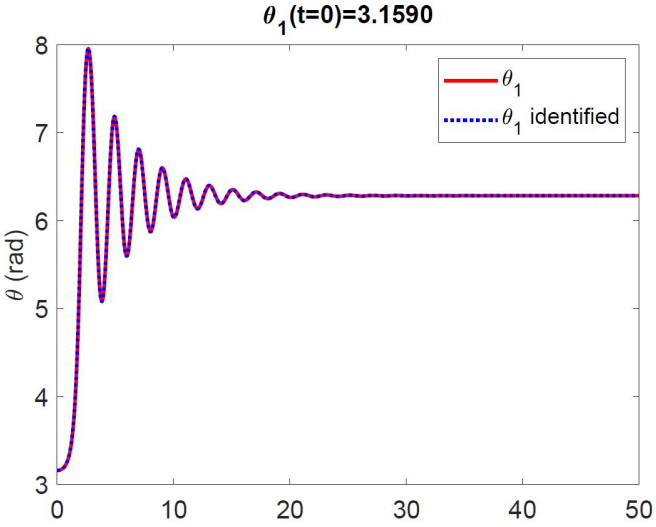


Figure 2.5: The SINDy algorithm correctly predicts the angle of a pendulum

The equation to discover is:

$$\dot{\theta} = \omega \tag{2.15}$$

$$\dot{\omega} = -9.81sin(\theta) - 0.5\omega \tag{2.16}$$

SINDy discovered the following equation:

$$\dot{\theta} = 1.0000\omega \tag{2.17}$$

$$\dot{\omega} = -9.8151sin(\theta) - 0.4983\omega \tag{2.18}$$

However if we allow the algorithm to build a bigger library by adding combinations between the polynomials and trigonometric function up to order 2 (e.g. $\omega * sin(\theta)$ or $sin^2(\theta)$) it overfits and produces an incorrect result while retaining high accuracy. Here it finds:

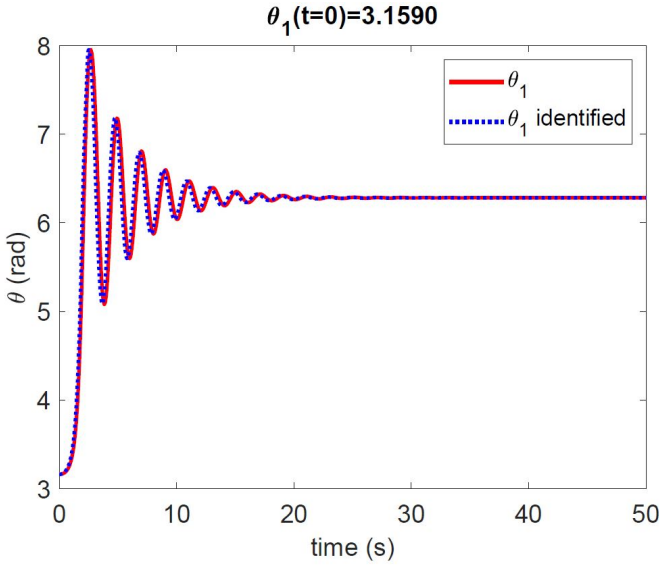


Figure 2.6: The SINDy algorithm almost predicts the angle of a pendulum correctly. The found equations are overfitted.

with:

$$\dot{\theta} = 102.5688 - 44.8708\theta + 0.9999\omega + 4.0989\theta^2 - 40.0247sin(\theta) - 13.4495cos(\theta) + 4.7156\theta sin(\theta) + 4.9329\theta cos(\theta) - 1.1973sin(\theta)cos(\theta) \quad (2.19)$$

$$\dot{\omega} = 89.4183 - 39.0261\theta - 0.5010\omega + 3.5596\theta^2 - 44.5361sin(\theta) - 11.4062cos(\theta) + 4.1027\theta sin(\theta) + 4.2443\theta cos(\theta) - 1.0008sin(\theta)cos(\omega) \quad (2.20)$$

We see that the algorithm is very susceptible to the library that is user-defined. There is no library that is suited for all formulas, as that would mean building an infinitely large library to ensure the needed combination is included in the library. However, even if this were possible, the algorithm would most likely overfit the problem, as we have seen with the single pendulum. The SINDy algorithm is therefore only recommended if the user has at least some knowledge of what kind of equation it is trying to find. For example, the SINDy algorithm would work well if you know that a polynomial describes the system. On the contrary, SINDy would be less convenient if you're trying to find an equation like $\sqrt{\frac{x^{2/3} * sin(x)}{y}}$, as you would need to input the exact equation in the library.

2.3.2. Genetic programming

A more general algorithm is Genetic programming. Here many random functions are composed and through an evolutionary algorithm, it proposes a function. As

an example, the toolbox gplearn is described [26]. Representing the formula in the form of a tree makes it easier to make sensible adaptations. In figure 2.7 such a representation is shown. At first, many random structures and values are generated to create many formulas. These formulas compete against each other in a tournament. In order to determine which formula performs best, its fitness is evaluated. The fitness is evaluated through an error metric that needs to be optimised, e.g., the mean absolute error.

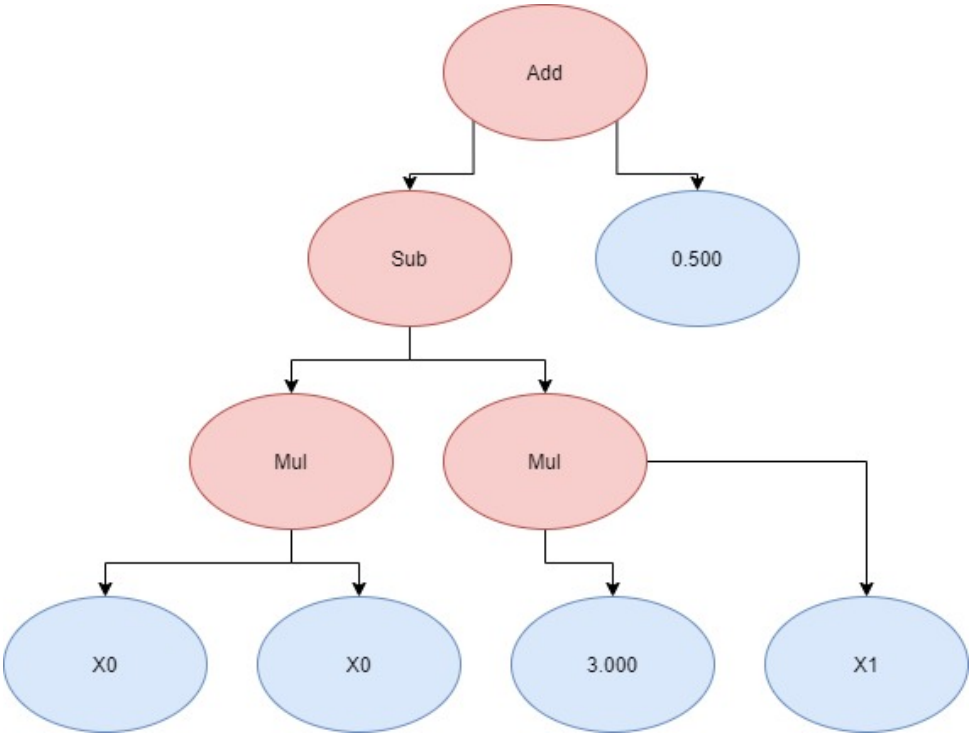


Figure 2.7: Representation of formula: $X_0 * X_0 - 3.000X1 + 0.500$ in genetic programming. Here "add", "sub" and ""mul" represent the operaters: addition, substraction and multiplication. "X0" and "X1" represent the variables of the equation.

Once the first generation of formulas is evaluated a new generation needs to be formed. There are four ways to create new formulas for the new generation. The first one is Crossover. Crossover mixes the formulas from the last generation by taking the winner of a tournament and selecting a random subtree to be replaced by a random subtree of a second tournament.

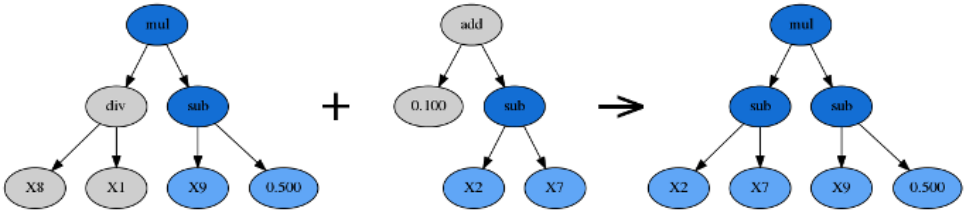


Figure 2.8: Example of crossover. Crossover mixes the formulas from the last generation by taking the winner of a tournament and selecting a random subtree to be replaced by a random subtree of a second tournament. Reprinted from gplearn, by T. Stephens, 2019, retrieved from https://gplearn.readthedocs.io/en/stable/intro.html

The second mutation method is subtree mutation. Subtree mutation mixes the formulas from the last generation by taking the winner of a tournament and selecting a random subtree to be replaced by a randomly generated subtree.

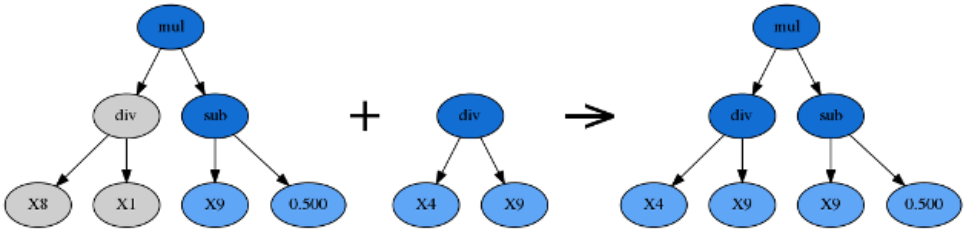


Figure 2.9: Example of subtree mutation. Subtree mutation mixes the formulas from the last generation by taking the winner of a tournament and selecting a random subtree to be replaced by a randomly generated subtree. Reprinted from gplearn, by T. Stephens, 2019, retrieved from https://gplearn.readthedocs.io/en/stable/intro.html

The third mutation method is hoist mutation. Hoist mutation fights bloat (a function with a large length) by removing a part of a random subtree of the winner of a tournament. This causes more parsimonious solutions. Bloat can also be controlled by penalising larger programs while evaluating the fitness of a function.

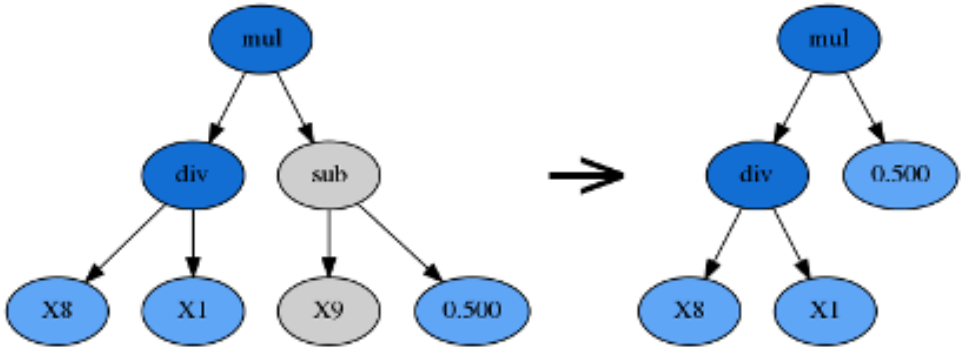


Figure 2.10: Example of hoist mutation. Hoist mutation fights bloat (a function with a large length) by removing a part of a random subtree of the winner of a tournament. Reprinted from gplearn, by T. Stephens, 2019, retrieved from https://gplearn.readthedocs.io/en/stable/intro.html

The fourth mutation, point mutation replaces a node of the winner of a tournament. This node is replaced with a node of the same arity if it is an operator or another constant if it is a constant.

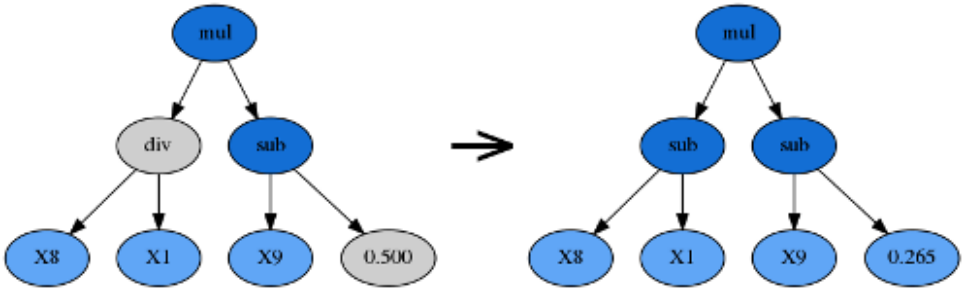


Figure 2.11: Example of point mutation. Point mutation, replaces a node of the winner of a tournament. Reprinted from gplearn, by T. Stephens, 2019, retrieved from https://gplearn.readthedocs.io/en/stable/intro.html

The algorithm terminates once either a set number of generations have evolved or once the stopping criteria are met.

2.3.3. AI Feynman

AI Feynman is a symbolic regression toolbox that relies on simplifying a dataset into smaller sub-problems. These sub-problems can then be solved using simple techniques, such as polynomial regression or a brute force search. The approach uses neural networks to identify simplifying properties in a dataset. Examples of simplifying properties are symmetry and separability. Symmetry checks if two variables

can be replaced with a single variable, i.e. can function $f(x, y)$ with two variables x and y be replaced with another function $g(z)$? Here variable z is a combination of variables x and y, e.g. $z = y - x$. Separability checks if a function can be described as a sum or product of two other functions, with the variables separated, i.e. can a function $f(x, y)$ be replaced with the product or sum of two different functions $g(x)$ and $h(x)$ like in $g(x) * h(y)$ or $g(x) + h(y)$?

The whole process is shown in figure 2.12. Dimensional analysis uses the units of the variables to simplify the equation by requiring the units of the two sides of an equation to match. The dataset is then reduced to a smaller one with fewer variables by replacing the variables with dimensionless ones. This step can also be skipped if no units are available. Then a neural network is trained to test for symmetry and separability. The neural network then simplifies the dataset through symmetry, separability and by equating the variables. Afterwards transformations are tried with the following functions: square root, raise to the power of 2, log, exp, inverse, sin, cos, tan, arcsin, arccos, and arctan. After each step the algorithm checks if the equation can be found with a polynomial fit or a brute force search.

2.3.4. Eliminating the need for the derivatives

One limitation for current methods of symbolic regression is that for a differential equation you would need the positions and its derivatives. By numerically calculating the derivatives, the error is increased so it would make sense to try to eliminate the need for the derivatives. The need for the derivatives can be eliminated if neural networks are used to fit data [11]. Based on the same idea, it is described in appendix A.1, how the need for the derivatives can be eliminated in symbolic regression as well. The SINDy algorithm, however, would need structural adaptations to do this. For genetic programming, this fits easier in the framework provided. In genetic programming, numerous random functions are tried and it outputs a vector of the derivatives. This vector is then compared to the derivatives acquired through differentiation. Instead, one can integrate the outputted vector of the derivatives and compare it to the input of the algorithm to see if the vector is satisfactory instead.

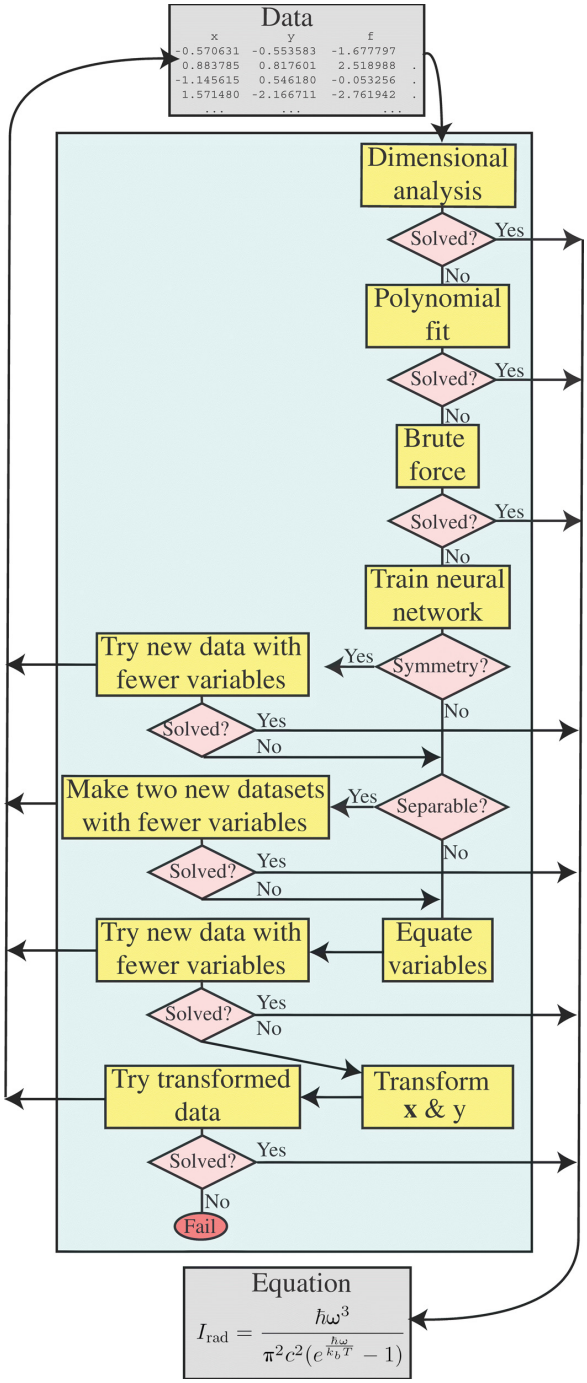


Figure 2.12: The discovery process of AI Feynman. Reprinted from "AI Feynman: A physics-inspired method for symbolic regression", by S.M. Udrescu et al., 2020, Science Advances, Volume 6 number 16

A difference is that in SINDy the derivatives are needed for the least-squares problem and it outputs a matrix Ξ that describes the function you want to find. In order to eliminate the need for a derivative SINDy needs to be changed. One can do this, for example, by changing the output to derivatives and obtaining Ξ through randomness. You then optimise the values for this matrix until you get values that are satisfactory according to a least-squares algorithm. These derivatives could also be integrated to compare to the positions in a least-squares problem.

2.3.5. Pareto frontier

In order to get a unique solution one could observe the Pareto frontier. An example of a Pareto frontier is shown in figure D.5. Here, at the corners of the front line, you cannot improve the predictive ability or decrease the number of nodes. One should then manually examine which formula is the correct one or use one of the validation techniques for variant and invariant equations.

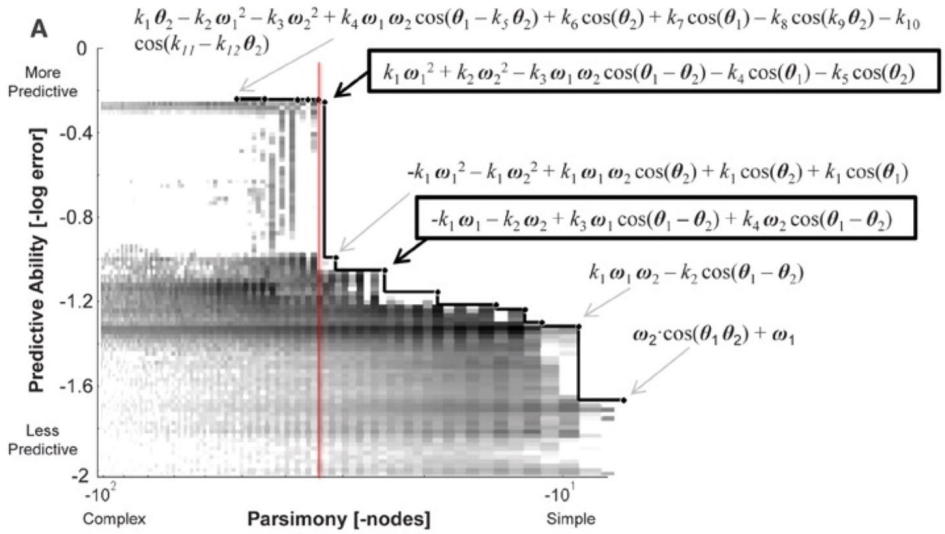


Figure 2.13: Example of the Pareto frontier for a double pendulum, Reprinted from "Distilling Free-Form Natural Laws from Experimental Data", by M. Schmidt et al., 2009, Science, Volume 324 Issue 5923, Pages 81-85, Copyright American Association for the Advancement of Science

2.3.6. Validation of variant and invariant equations

In figures 2.14 and 2.15 methods are shown to validate variant and invariant equations respectively [8, 9]. Validation is important in case you are not able to identify the correct equation out of multiple proposed ones. The idea described in figure 2.14 is to find test data where the predictions of the found equations disagree with each other. New iterations of equations are then trained on the new test data. If this equation agrees with one of the earlier equations it is validated.

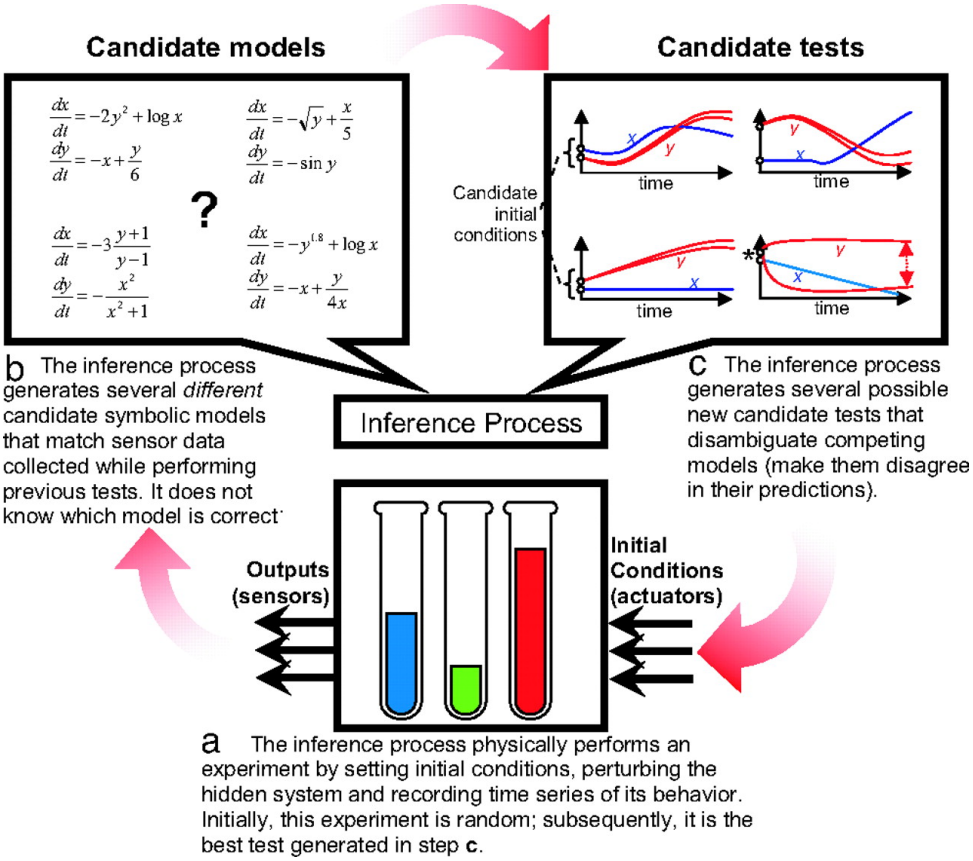


Figure 2.14: Validation for variant equations, Reprinted from "Automated reverse engineering of nonlinear dynamical systems", by J. Bongard et al., 2007, PNAS, Volume 104, no. 24, Pages 9943-9948

Validation is tougher for invariant equation because infinitely many invariant equations can be found from data that are correct over the entire range of data. For example $x_1 + 0.543 - x_1$ or $\frac{sin^2(x_1)+cos^2(x_1)}{x_2} - \frac{x_1}{x_2}$ are both invariant but meaningless for physical systems. Furthermore, there exist infinitely more solutions with a very small error, such as $\frac{x_1}{10^6} + 4.32$. In order to tackle this, the procedure from figure 2.15 is proposed [8]. Here the partial derivatives that are calculated from the data are compared to the symbolic partial derivative of the candidate functions. If these are satisfactory it is validated and otherwise, it loops to make new candidate functions.

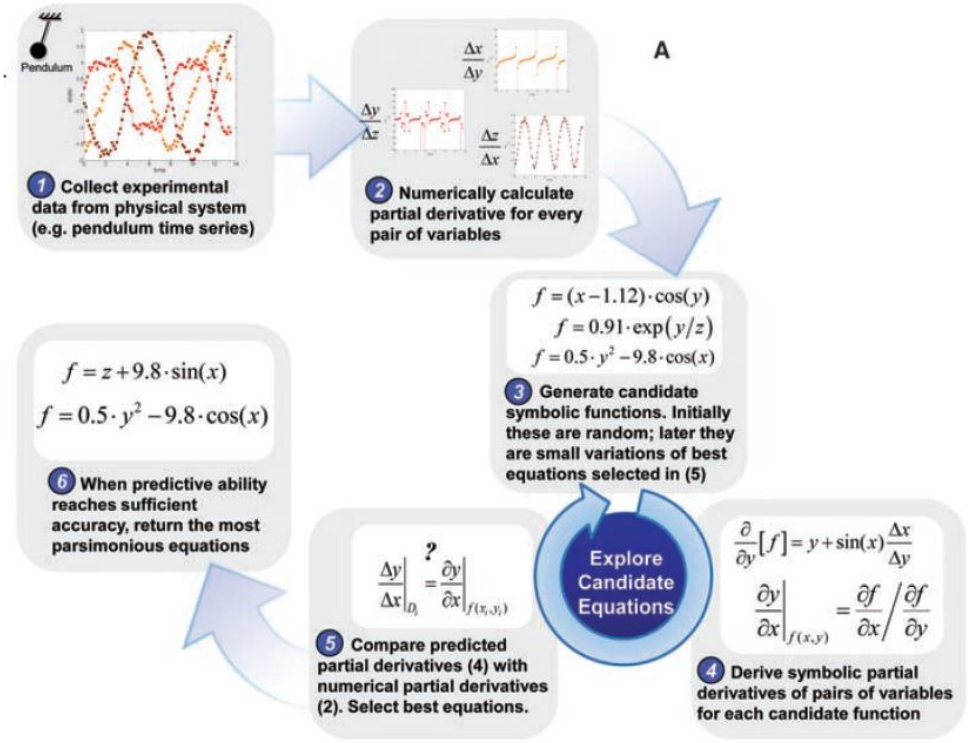


Figure 2.15: Validation for invariant equations, Reprinted from "Distilling Free-Form Natural Laws from Experimental Data", by M. Schmidt et al., 2009, Science, Volume 324 Issue 5923, Pages 81-85, Copyright American Association for the Advancement of Science

3

Research question, objectives and research methodology

3.1. Research objectives and questions

The amount of studies that pursue exact symbolic regression is still at an early stage. Often times the aim is to pursue a good quality fit but the discovered equations have no physical basis and are not necessarily the governing equation of the system. The few studies that do pursue finding the equations with a physical basis have promising results with synthetic data with little to no noise but often have insufficient capabilities to handle experimental data. The aim of this research is to bridge the gap of capabilities on clean data and noisy data.

There are many non-symbolic regression methods that provide a good fit even when the data is noisy. One method that seems particularly suited for this, is Gaussian process regression. Gaussian processes' universal approximating and smoothening capabilities make it an effective tool for noise-reduction in data. The combination of symbolic regression and the smoothening abilities of Gaussian processes have not been explored in research so far.

In this research it is therefore proposed to split the symbolic regression process into a two-step process. The first part will focus on transforming the properties of the data to the properties of the clean synthetic data that are used in the studies for which current symbolic regression methods have promising results. The second step is actually doing the symbolic regression.

In summary, the objective of this research is to explore the effectiveness of using Gaussian processes as a noise-reduction tool, in combination with symbolic regression.

In order to reach this objective, the following main research questions are formu-

lated:

- How can Gaussian processes be used to minimise the effects of noise in symbolic regression?
- How can the derivatives of a dynamical system be robustly and numerically calculated?
- How can machine learning tools be used to get consistent results in symbolic regression?

3.2. Research methodology

In this research, we studied how machine learning techniques can be combined with symbolic regression to overcome each other's shortcomings. The idea is to find governing equations, in situations where only samples of the dynamical behaviour of a system are available, without needing prior knowledge of the system. In order to provide a general framework, a numerical package is developed to automatically create and preprocess noisy dynamical data in a consistent manner. Here, two approaches are investigated. The first approach is for a case where both the positions and its derivatives can be measured. The second approach is for a case were only the positions can be measured. In order to get the most out of the noise-reduction step, the effect of different kernels in Gaussian processes are investigated. Furthermore, we studied how symbolic regressors can be optimised to find consistent results, without being too dependent on an arbitrary choice of hyperparameters. The dynamical test systems exhibit different behaviour, such as being forced or autonomous and having different periodicity or chaos. One such system, the duffing oscillator, can exhibit different periodicity and even chaos. We investigate if for each case the same equation can be found. The test systems are subjected to different levels of noise to get an indication of the maximum allowable noise and sensitivity for different dynamical systems. We finally test if the obtained relations still retain all dynamical properties.

II

Paper

4

Paper: Data-driven techniques for finding governing equations of nonlinear dynamical systems

Data-driven techniques for finding governing equations of noisy nonlinear dynamical systems

H.C. Lingmont
Precision and Microsystems Engineering
Delft University of Technology
2628 CD Delft, The Netherlands
hidde.lingmont@hotmail.com

Dr. F. Alijani
Precision and Microsystems Engineering
Delft University of Technology
2628 CD Delft, The Netherlands
f.alijani@tudelft.nl

Dr. M.A. Bessa
Materials Science and Engineering
Delft University of Technology
2628 CD Delft, The Netherlands
M.A.Bessa@tudelft.nl

***Abstract*—The advent of machine learning and the availability of big data brought a novel approach for researchers to discover fundamental laws of motion. Computers allow to quickly find underlying physical laws from experimental data, without having in-depth knowledge of the system. Applications are widespread among numerous fields such as physics, chemistry, engineering, biology, climate science and finance. However, the problem is that often experimental data are polluted with noise. This causes symbolic regressors to capture the noise but not the underlying physics. In this work, we leverage the noise reduction properties of non-parametric machine learning, to improve symbolic regression methods. This combination allows for numerically robust differentiation and significantly increases the noise tolerance of common symbolic regressors. A new strategy is exemplified by combining Gaussian processes and the symbolic regression toolbox SINDy for finding governing equations from data. The method balances the quality of the fit and parsimony to avoid overfitting. The method is tested on well-known dynamical systems with varying properties. These will include the Duffing oscillator, as an example of a forced system, the Van der Pol oscillator, as an example of an autonomous system and the Lorenz attractor as an example of a chaotic system.**

***Index Terms*—Symbolic regression, Gaussian process regression, numerically robust differentiation, Machine learning, optimization, dynamical systems**

I. Introduction

In the field of system identification, many advances have been made to discover mathematical models from data [1]. In recent years, the role of computers in finding these models, instead of researchers based on their understanding of physics, has been increasing. A powerful data-driven method for creating interpretive models is symbolic regression.

Unfortunately, the field of symbolic regression is not very extensive. An extra challenge is that not all of these studies are interested in finding an equation with a physical basis but rather just an equation that provides a good fit, for example in forecasting the stock market [2]. Recently studies have been carried out with the aim of finding the governing equation of a system [3, 4, 5, 6, 7]. However, these studies still provide limited usability as experimental data are often polluted with noise and therefore much more challenging to work with than the relatively clean synthetic data provided in the studies. In order to make these algorithms more useful for experimental data, it is important to make them more robust to noise. Furthermore, these algorithms can discover multiple equations depending on the hyperparameters. This makes it hard to select a single equation as the governing one.

The advent of machine learning allows for new approaches. These approaches rely on self-learning algorithms that can accurately capture the dynamics of a system. These include: Neural networks [12, 13], recurrent neural networks [14], radial basis functions [8], the Nonlinear AutoRegressive Moving Average with eXogenous input (NARMAX) model [9, 10, 11], and Gaussian Processes [15, 16, 17, 18, 19]. These techniques can be tuned to be more robust to noise than symbolic regression methods and can be used for interpolation, finding patterns and extracting dynamical properties. However, no physical understanding of the system is gained and often these methods score poorly on extrapolated data. Instead, it would be better to know the governing equations of the system.

Consider the situation where a researcher wants to discover what equations govern the Lorenz attractor. If the researcher attempts to achieve this using common symbolic regressors, he or she will find multiple equations depending on the hyperparameters of the symbolic regressor. If the data are relatively noise-free, the governing equation might be on that list. In practice, the data obtained by the researcher contains too much noise, so multiple equations are discovered that seem to have a good fit for interpolation but fail to extrapolate well. In these cases the governing equation was not found. The researcher needs a more noise-robust algorithm that is able to find the equation.

Therefore the objective of this research is to investigate if symbolic regression can be made more robust to noise by adding a noise reduction step before it is attempted to find an equation. This is done by combining the accurate fitting techniques of machine learning with the interpretability of symbolic regression.

II. SymbolicAI

In this work, a new hybrid symbolic regression and machine learning method is proposed, namely SymbolicAI. The idea is

to split the process in three steps. The first step is to use machine learning techniques for noise reduction. In this work Gaussian processes are used to exemplify the potential of this method but other techniques, such as Artifical Neural Network (ANN) regression or splines can be used. These regression methods are able to predict new smoothened data but provide no interpretability. Here a distinction is made between a scenario where the derivatives are available and a scenario where no derivatives are available. In the latter case, Gaussian processes can be used to minimise the accumulation of error in numerically differentiated data as well. In the second step, we use the data predicted by the Gaussian process for a symbolic regressor. Here the SINDy algorithm [3, 20] is chosen to find equations from data. However, other symbolic regressors, such as genetic programming [4, 5] or AI Feynman [6], could be used for this step as well. Depending on the hyperparameters of the SINDy algorithm, multiple equations can be found. In the third step, a new algorithm is proposed that balances equation length with the performance of the fit on extrapolated data. This provides a consistent manner of selecting an equation from multiple ones.

III. Gaussian processes

Gaussian process regression is a strong regression tool that can be used to transform data unsuitable for symbolic regression into data with more preferable qualities. It is a Bayesian approach for non-parametric regression, where many samples are taken from a high-dimensional multivariate Gaussian distribution. A Gaussian process is defined as follows:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}')) \tag{1}$$

This equation describes that a function $f(\mathbf{x})$ is sampled on the basis of data $\mathbf{x}$, from a Gaussian process distribution $\mathcal{GP}$ with mean $m(\mathbf{x})$ and covariance $K(\mathbf{x}, \mathbf{x}')$. Subsequently, the average and variance of many samples are taken, which acts as the prediction line, with a confidence interval. A more exhaustive review of Gaussian processes can be found in appendix B.

The way the covariance matrix is calculated is defined through its kernel function. The performance of Gaussian processes depends heavily on the choice of the kernel. Finding the right kernel is a non-trivial task, as often the adequacy of the choice relies on the experience of the researcher. In appendix B.1.1 rules of thumb for constructing a good fitting kernel are described, to aid in this process. The search for the right kernel can also be automated. This process is described in appendix B.1.2. This can be helpful when one is not able to find a good fitting kernel through intuition but has the downside of being computationally costly.

There are several advantages in using Gaussian processes over other regression methods. In subsection III-A it is described how a common problem in parametric regression is tackled, namely that the form of the fit needs to be known. Another problem with other regression methods, such as neural networks, is that a large number of points is required to get a good fit. In subsection III-B it is described how this problem can be tackled with Gaussian processes and how Gaussian processes still work if the data are not equally distributed over time. In subsections III-C and III-D it is described how Gaussian processes are effective even in the presence of noise and how Gaussian processes provide a confidence interval to show the reliability of the fit. This addresses two other common problems in regression, namely that noise causes the regression method to be prone to errors and that the user has no indication of how reliable the fit is. Lastly, in subsection III-E it is described how Gaussian processes provide a noise-robust way of obtaining a signal's derivative.

A. non-parametric regression

There are many ways to fit a line through points. Many methods require the researcher to look at the data and assume the form of the line. If the form is not known, one can switch to non-parametric regression. This can take on many forms and thus there is no need to guess the form. Gaussian processes are a form of non-parametric regression.

B. Addressing irregularly sampled, scarcity and sparsity in data

Gaussian processes can also be useful in combatting irregularly sampled data. Often for symbolic regression, there needs to be a regular time interval between the data points. Gaussian processes do not need this and can thus be used to construct a regression model. Once a model is established, the model can be used to predict data points with a constant distance in time. These data points can then in turn be used for symbolic regression. A similar approach can be taken if there are too little data points for a symbolic regressor to work. Gaussian processes are known to be effective regressors, even if there are relatively few data points. Once a model is constructed, it is possible to generate more data points from the distribution to use for the symbolic regressor.

C. Addressing noise and overfitting

Gaussian processes often smoothen a dataset as it returns the mean of a distribution of functions. This makes it effective in finding long-term trends, periodicity, variations of different amplitudes and helps prevent overfitting. White noise has the property of varying with a very small timescale and having no accumulating trend. To detect this, one could equip a squared exponential kernel with an infinitely small width to account for these local variations. Through modularity of a kernel function, it is possible to construct a kernel that accounts for all the variations of a function and adding a white noise kernel (A squared exponential with a width of near-zero) to account for the noise. This way Gaussian processes can effectively reduce the noise on a dataset. This approach becomes harder for non-white noise, as there is another trend that emerges from the noise as well. Often it becomes very tough to distinguish between the trend from the noise and the trend from the signal.

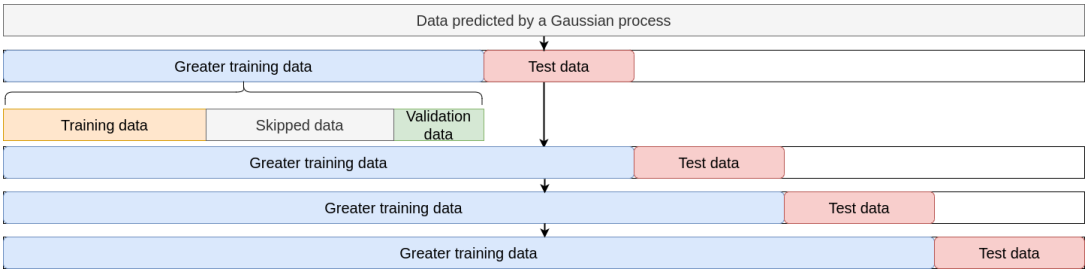


Fig. 1: The data is split into training, validation and test data. SINDy is trained with different hyperparameters on the training data and an equation is chosen with the validation data. The performance is then tested through rolling cross-validation with the test data.

D. Providing uncertainty estimates

Due to the fact that Gaussian processes provide a distribution over a function, it is equivalent to generating many sample functions from it. These sample functions could then be used to calculate a mean function and the variance for a confidence interval. These can be calculated using the following formulas:

$$\boldsymbol{\mu_M} = \mathbf{K}_{MN} \left(\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N\right)^{-1} \mathbf{y} \tag{2}$$

$$\Sigma_M = \mathbf{K}_M - \mathbf{K}_{MN} \left(\mathbf{K}_N + \sigma_n^2 \mathbf{I}_N\right)^{-1} \mathbf{K}_{MN}^T \tag{3}$$

For given data $\mathbf{y}$ (zero-mean), the mean $\boldsymbol{\mu_M}$ and variance Σ_M are calculated at M unseen points. These are calculated with the kernel matrix $K = \begin{bmatrix} K_M & K_{MN} \\ K_{MN}^T & K_N \end{bmatrix}$, where N denote the training points. The variance of the noise is given by σ_n. The confidence interval gives an idea of how accurate the prediction is and thus it provides the researcher with uncertainty estimates.

E. Access to the derivatives

Once a function is smoothened its numerical differentiation can be more robustly performed. If the chosen kernel is differentiable, the derivative of a Gaussian process is another Gaussian process. From this differentiated Gaussian process a new mean and variance can be obtained. This process is elaborated in appendix C. This can be very useful if one cannot measure the derivatives directly. In many cases, one only has access to noisy positions. In order to find a governing differential equation, access to its derivatives needs to be available as well. However, with common numerical differentiators the noise is greatly amplified. Gaussian processes provide a smooth basis to differentiate from. The derivative can then be found by either differentiating the Gaussian process or by using another numerical differentiator with data predicted by the Gaussian process.

IV. Symbolic regression

In order to find symbolic relations between data, the SINDy algorithm [20] is considered. The fundamentals of SINDy are described in section 2.3.1. For this method, it is important to have an idea of what terms make up the equation. A library of these proposed terms can be constructed, which is, together with the data, the input. The algorithm then finds a combination of candidate functions that best describe the data. It punishes larger equations in its scoring as a prevention method for overfitting. However, the regularisation parameters that control this are not optimised. With different values for the hyperparameters, many different equations can be found. In order to automatically find good hyperparameters the following optimisation process is proposed in the SymbolicAI toolbox:

The data are split into training, validation and test data, as shown in figure 1. The training data are used to generate equations with SINDy and the validation data are used to choose an equation. Data between the training data and validation data are skipped because the predictive abilities of wrong equations usually become worse the further away from the training data is extrapolated, which makes it easier to single out the correct equation. The obtained equation is then tested through rolling cross-validation, to give an estimate of its accuracy.

Using the coefficient of determination and the length of the equation, a Pareto frontier can be created. An example of a Pareto frontier is shown in figure 3. In order to decide between the equations, L1-regularization can be used. This is shown in equation 4, where λ is the regularisation parameter. It was found that often only equations with the correct terms score high. However, sometimes equations that add small extra terms that fit the noise, score a little bit higher. In order to prevent these extra near-zero terms, it is recommended to keep $\lambda = 0.1$ and thus punish longer equations.

$$score = R^2 - \lambda * length \tag{4}$$

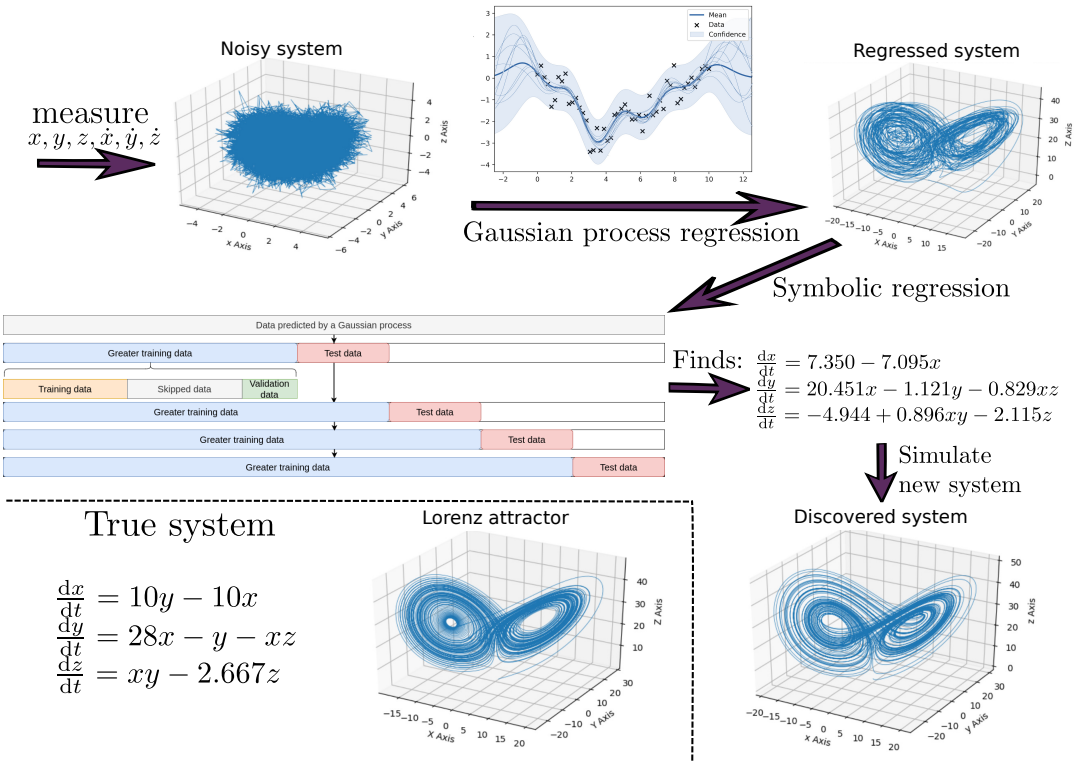


Fig. 2: Process approach 1

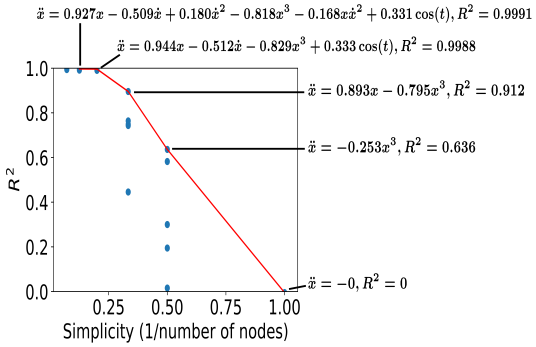


Fig. 3: Pareto frontier, each dot represents an equation obtained with SINDy. The simplicity is defined as $\frac{1}{numberofnodes}$ where each operation and variable in an equation is a node.

V. Method

Experimental data may contain excessive amounts of noise that make it unsuitable for direct symbolic regression. In order to emulate this, data are generated from well known examples, where the amount of noise is controlled. The data are generated by integrating initial conditions according to a known equation over a specified time span. This provides clean data, which can be used as a reference versus noisy data. To emulate experimental data, noise is added to the clean data. In order to compare the noise removal abilities of the algorithm, the amount of noise should be added in a consistent manner. To do this, the clean data are first standardised to be zero-mean and have a variance of 1. Then noise is added that is also zero-mean and with a consistent variance for all equations.

There are two scenarios for finding governing equations from data that will be examined next. The first is for noisy data, with access to the derivatives and in the second only the positions are available.

Scenario 1: noisy data with access to the derivatives
To reduce the amount of noise, the aim is to predict new smoothened data from a Gaussian process model. In many cases, it is useful to standardise data before performing Gaus-

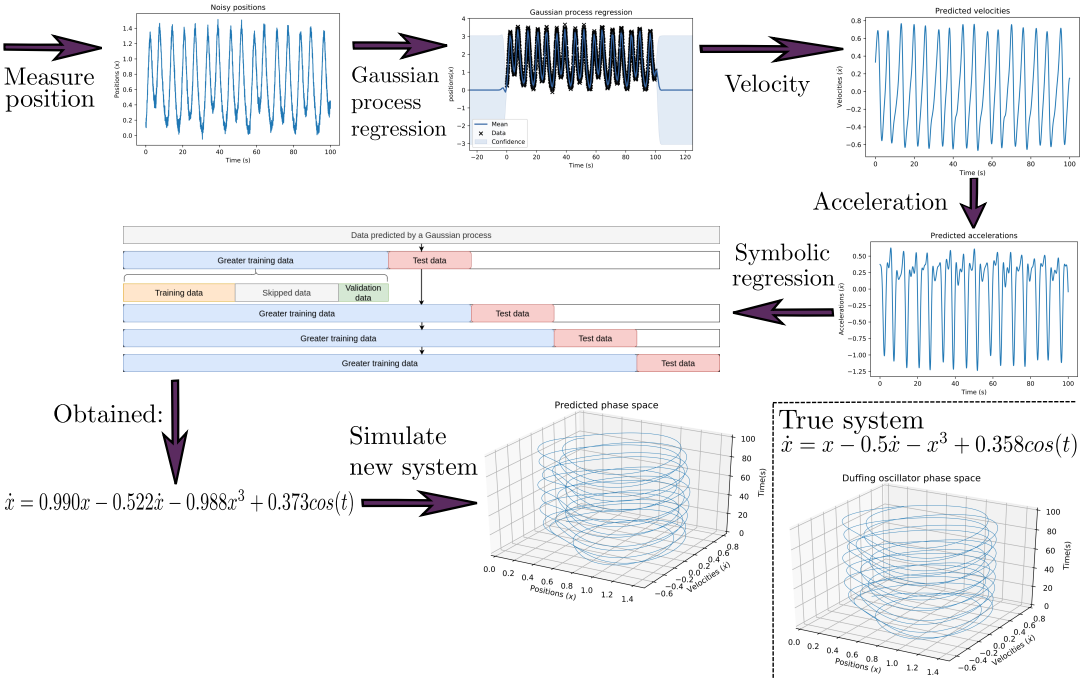


Fig. 4: Process approach 2

sian process regression for the following reasons:

- In the regression process it is assumed that the output is zero-mean. Standardising the data beforehand justifies this assumption.
- Scaled data are more suitable for hyperparameter estimation of the kernel function.
- as can be seen in equations 2 and 3, the kernel matrix needs to be inverted. This can be an ill-conditioned matrix if the data are not scaled properly.

The data are split into training and test data to obtain an idea of how accurate the regression was. The predicted data are then used to find an equation using the process described in section IV. The whole process is shown in figure 2.

Scenario 2: noisy data without access to the derivatives
In this scenario, only the noisy positions are available. Now the Gaussian process is used to fit only the standardised positions. The velocities and accelerations are then obtained by taking the derivative of the Gaussian process. The predicted data can then be used to find an equation using the process described in section IV. The whole process is shown in figure 4.

In order to provide a generalised framework, SymbolicAI includes tools to do all the preprocessing and regression consistently, including an automated kernel search. A description of the package can be found in appendix B.2.

VI. Test problems

In order to test the algorithms, multiple well-studied problems have been chosen from the field of nonlinear dynamics. Three examples will be shown here and other examples can be found in appendix E.2. The field of nonlinear dynamics provides challenging equations for smootheners. The data tends to oscillate quickly, making it hard to distinguish from noise and hard to fit with a smooth line.

For each of the examples, data are generated from the provided equations and initial conditions.

Van der Pol oscillator
The Van der Pol oscillator is an example of an autonomous nonlinear dynamical system. The system converges into a limit cycle and exhibits period 1 motion. The system can be described by the following equation:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0 \tag{5}$$

The data are generated with $\mu = 2.5$, which is dimensionless. The Van der Pol oscillator is a non-

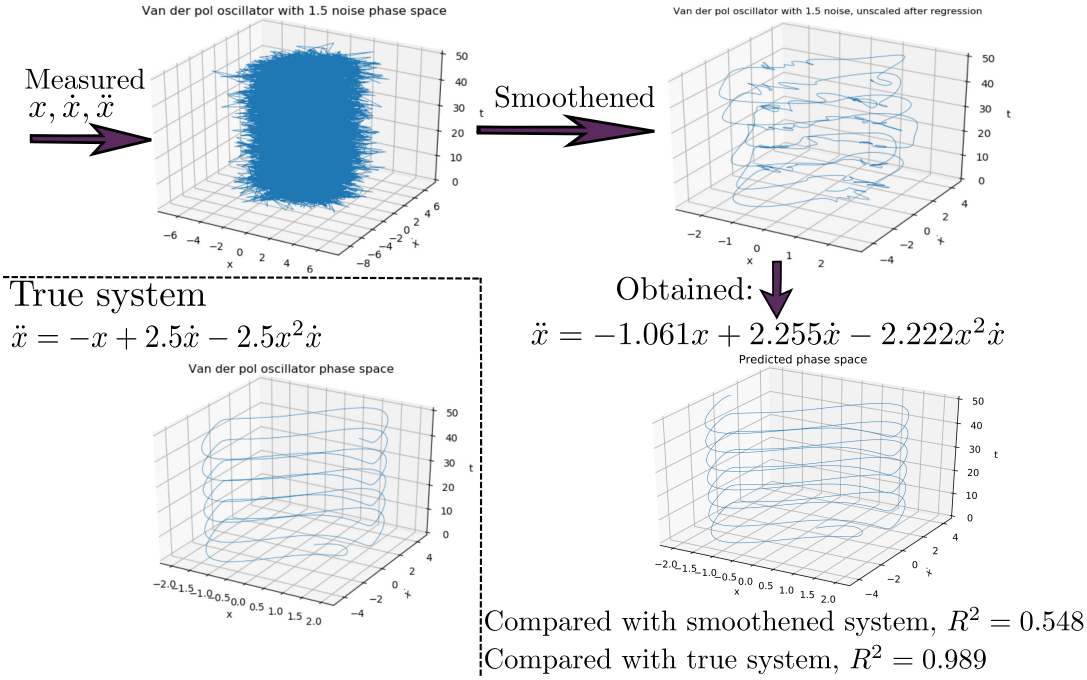


Fig. 5: Result Van der Pol oscillator with added noise that has a variance that is 1.5 times bigger than the signal's variance

conservative oscillator with nonlinear damping.

Duffing oscillator

The Duffing oscillator is an example of a forced nonlinear dynamical system that can exhibit very different dynamical behaviour. The duffing equation is described by:

$$\ddot{x} + 2\mu\dot{x} + kx + \alpha x^3 = F\cos(\omega t) \tag{6}$$

Here x is the displacement, μ the damping coefficient, k the spring constant, α the nonlinear stiffness, F the amplitude of the periodic driving force and ω the angular frequency of the periodic driving force. The data are generated with the following parameters: $\mu = 0.25\,\mathrm{s}^{-1}$, $\alpha = 1\,\mathrm{s}^{-2}\mathrm{m}^{-2}$, $k = -1\,\mathrm{s}^{-2}$, $\omega = 1\,\mathrm{rad s}^{-1}$ and initial conditions $x(0) = 0.09\,\mathrm{m}$ and $\dot{x}(0) = 0\,\mathrm{ms}^{-1}$. Given different values for the forcing parameter F, the equation exhibits a period-doubling route to chaos. The system will have a period 1 solution (one dominant frequency) for $F = 0.325\,\mathrm{ms}^{-2}$ and below. For $F = 0.348\,\mathrm{ms}^{-2}$ and $F = 0.349\,\mathrm{ms}^{-2}$ it has period 2 motion, for $F = 0.357\,\mathrm{ms}^{-2}$ period 4, for $F = 0.358\,\mathrm{ms}^{-2}$ period 8 and for $F = 0.4\,\mathrm{ms}^{-2}$ chaotic oscillations. It is tested if the same equation can be found for all these conditions.

Lorenz attractor

The Lorenz attractor is a system of nonlinear dynamical equations that exhibit chaotic solutions for certain parameter values and initial conditions. The equation is used to model lasers [21], dynamos [22], electric circuits [23] and chemical reactions [24]. The system can be described by the following equations:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \sigma(y - x) \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= x(\rho - z) - y \\ \frac{\mathrm{d}z}{\mathrm{d}t} &= xy - \beta z \end{aligned} \tag{7}$$

Here the constants are $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$, which are dimensionless.

VII. Results

A. Examples with derivative information

In this scenario, noise is added to all data available. This means that for the Van der Pol oscillator and Duffing oscillator, noise is added to the positions, velocities and accelerations. In the case of the Lorenz attractor, the noise is added to positions and velocities. It was found that in every function tested, doing Gaussian process regression beforehand improves the obtained equation. If there was little noise, the coefficients would be more accurate. If there was more noise, it could still find the correct terms of the equation, whereas

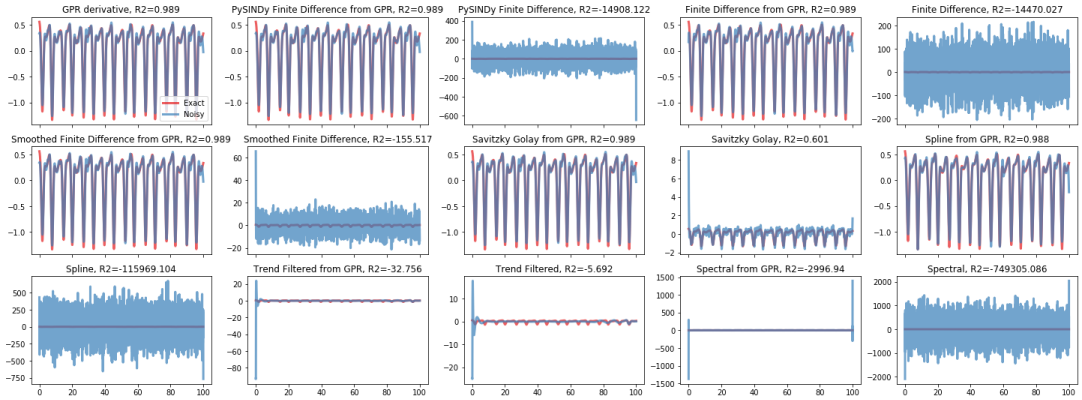


Fig. 6: Comparison of the second derivative of the Duffing oscillator with 10% added noise relative to the magnitude of the clean signal. The red lines are the true values for the second derivative and the blue lines are the values predicted by the method described in the subtitle. If from GPR is added to the subtitle, the method worked with positions predicted by a Gaussian process. Otherwise unprocessed noisy positions are used.

this would not be possible if no Gaussian process regression was done beforehand. In figure 5 the result is shown for a Van der Pol oscillator with 150% added noise. It can be seen that Gaussian processes can greatly reduce the error due to noise. Sindy and the hyperparameter optimisation then correctly find the equation. For the Gaussian process regression 9000 points were used. Increasing the number of points used for Gaussian process regression increases the noise reduction capabilities but the computation also scales with $O(n^3)$, with regard to the number of points.

The results for all test cases are shown in table I. Here 2000 points are used for the Gaussian process regression. With the Gaussian process model, 60,000 points are then predicted in the same time-interval. The result is compared to doing symbolic regression without first regressing with Gaussian processes. In order to compare the increased robustness to noise, the same amount of points (60,000) are used for the symbolic regression without Gaussian processes.

B. Examples without derivative information

In this scenario, a situation is simulated where no derivative information is available. Here Gaussian process regression is applied to the positional data. The derivatives are then numerically calculated from the smoothened positions.

In order to compare the effectiveness of using the derivative of a Gaussian process, the result is compared to common other differentiators. The other differentiators have been tested for two cases. In the first case, the methods work with raw data from the noisy signal and in the second case, the methods work with data predicted by a Gaussian process. In case the kernel is not differentiable, we cannot take the derivative of the Gaussian process but we can still get a robust approximation of the derivative by using another differentiation method with positional data predicted by the Gaussian process. In figure 6, the approximations of the second derivative of the Duffing oscillator with 10% added noise, relative to the magnitude of the clean signal, are compared. It can be seen that using data predicted by a Gaussian process works best. If more noise is added, it can be seen that taking the derivative of the Gaussian process is the most robust, but using finite difference provides a quick and robust alternative, given that data predicted by a Gaussian process are used. Using data predicted from Gaussian processes sharply improves the accuracy of all methods except for the trend filtered one. Since the trend filter also filters the data, extra information of the signal might get lost, which explains the slightly lower score for the trend filtered prediction from the Gaussian process data.

The results for all test cases are shown in table II. Here 2000 points are used for the Gaussian process regression. With the Gaussian process model, 60,000 points are then predicted in the same time-interval. The test cases are tested with 5% and 10% added noise, relative to the magnitude of the signal.

C. Discovering equations from chaos

In order to test the robustness to chaos, the Duffing oscillator with a forcing $F = 0.4\,\mathrm{ms}^{-2}$ is evaluated at different time-spans. This exposes the algorithm to completely different behaviour that is governed by the same equation. It was found that it is possible to find the same equation in each interval. However, larger time-spans were required than for non-chaotic data of the Duffing oscillator. It is postulated that this is due to a larger pool of candidate functions. A downside of a wider time-span is that with the same amount of points the Gaussian process regression will perform worse. If more points

TABLE I: Results for all test problems with derivative information, with different amounts of noise. The result is compared to an approach were no Gaussian processes regression(GPR) is used. Percentages are relative to the magnitude of the clean signal. R^2_{True} is the R^2 compared to the true system and R^2_{GPR} is the R^2 compared to the smoothened system.

Functions	**5% noise added**	**15% noise added**	**50% noise added**
True Van der pol oscillator, $\ddot{x} = -x + 2.5\dot{x} - 2.5^2\dot{x}$			
With Gaussian processes	$\ddot{x} = -1.000x + 2.477\dot{x} - 2.489x^2\dot{x}$, $R^2_{True} = 0.99996$, $R^2_{GPR} = 0.997$	$\ddot{x} = -0.970x + 2.450\dot{x} - 2.429x^2\dot{x}$, $R^2_{True} = 0.9991$, $R^2_{GPR} = 0.976$	$\ddot{x} = -0.987x + 2.078\dot{x} - 1.967x^2\dot{x}$, $R^2_{True} = 0.960$, $R^2_{GPR} = 0.800$
Without Gaussian processes	$\ddot{x} = -0.997x + 2.437\dot{x} - 2.415x^2\dot{x}$, $R^2_{True} = 0.9989$, $R^2_{GPR} = 0.972$	$\ddot{x} = -0.974x + 2.071\dot{x} - 1.896x^2\dot{x}$, $R^2_{True} = 0.946$, $R^2_{GPR} = 0.796$	$\ddot{x} = -0.854x + 0.849\dot{x} - 0.398x^2\dot{x} - 0.4cos(t)$, $R^2_{True} = 0.360$, $R^2_{GPR} = 0.210$
True Lorenz attractor, $\dot{x} = 10y - 10x$ $\dot{y} = 28x - y - xz$ $\dot{z} = xy - 2.667z$			
With Gaussian processes	$\dot{x} = 9.880y - 9.893x$ $\dot{y} = 28.074x - 0.959y - 1.004xz$, $\dot{z} = 0.994xy - 2.645z$ $R^2_{True} = 0.99992$, $R^2_{GPR} = 0.995$	$\dot{x} = 9.246y - 9.267x + 0.403$ $\dot{y} = 26.578x - 0.454y - 0.972xz$, $\dot{z} = 0.975xy - 2.573z$ $R^2_{True} = 0.998$, $R^2_{GPR} = 0.973$	$\dot{x} = 6.349y - 6.295x + 1.302$ $\dot{y} = 19.020x + 1.299y - 0.777xz + 7.771 - 0.393z$, $\dot{z} = 0.892xy - 1.915z - 6.573 + 0.317x$ $R^2_{True} = 0.940$, $R^2_{GPR} = 0.783$
Without Gaussian processes	$\dot{x} = 9.744y - 9.717x$ $\dot{y} = 27.303x - 0.775y - 0.984xz$, $\dot{z} = 0.995xy - 2.6449z$ $R^2_{True} = 0.9997$, $R^2_{GPR} = 0.988$	$\dot{x} = 6.791y - 1.864x + 5.639 - 0.229t - 0.158xz$ $\dot{y} = 22.999x + 0.387y - 0.881xz - 3.803 + 0.159t$, $\dot{z} = 0.952xy - 2.456z - 1.830$ $R^2_{True} = 0.769$, $R^2_{GPR} = 0.847$	$\dot{x} = 2.914y - 2.174x + 3.683$ $\dot{y} = 12.112x + 1.156y - 0.533xz + 9.352 - 0.586z$, $\dot{z} = 0.615xy - 1.151z - 7.894 + 0.727x$ $R^2_{True} = 0.722$, $R^2_{GPR} = 0.461$
True Duffing oscillator with F=0.325 $\ddot{x} = x - 0.5\dot{x} - x^3 + 0.325cos(t)$			
With Gaussian processes	$\ddot{x} = 0.995x - 0.493\dot{x} - 0.996x^3 + 0.324cos(t)$, $R^2_{True} = 0.99997$, $R^2_{GPR} = 0.9996$	$\ddot{x} = 0.984x - 0.479\dot{x} - 0.985x^3 + 0.320cos(t)$, $R^2_{True} = 0.9997$, $R^2_{GPR} = 0.997$	$\ddot{x} = 0.936x - 0.419\dot{x} - 0.940x^3 + 0.302cos(t)$, $R^2_{True} = 0.995$, $R^2_{GPR} = 0.967$
Without Gaussian processes	$\ddot{x} = 0.987x - 0.488\dot{x} - 0.986x^3 + 0.317cos(t)$, $R^2_{True} = 0.997$, $R^2_{GPR} = 0.990$	$\ddot{x} = 0.901x - 0.403\dot{x} - 0.899x^3 + 0.265cos(t)$, $R^2_{True} = 0.984$, $R^2_{GPR} = 0.933$	$\ddot{x} = -0.220x^3 + 0.323cos(x)$, $R^2_{True} = 0.697$, $R^2_{GPR} = 0.563$
True Duffing oscillator with F=0.348, $\ddot{x} = x - 0.5\dot{x} - x^3 + 0.348cos(t)$			
With Gaussian processes	$\ddot{x} = 0.998x - 0.497\dot{x} - 0.999x^3 + 0.349cos(t)$, $R^2_{True} = 0.99998$, $R^2_{GPR} = 0.9996$	$\ddot{x} = 0.994x - 0.489\dot{x} - 0.997x^3 + 0.350cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.997$	$\ddot{x} = 0.972x - 0.451\dot{x} - 0.985x^3 + 0.346cos(t)$, $R^2_{True} = 0.998$, $R^2_{GPR} = 0.973$
Without Gaussian processes	$\ddot{x} = 0.988x - 0.489\dot{x} - 0.988x^3 + 0.341cos(t)$, $R^2_{True} = 0.9996$, $R^2_{GPR} = 0.991$	$\ddot{x} = 0.909x - 0.414\dot{x} - 0.906x^3 + 0.299cos(t)$, $R^2_{True} = 0.989$, $R^2_{GPR} = 0.920$	$\ddot{x} = -0.321x^2 + 0.377cos(x)$, $R^2_{True} = 0.717$, $R^2_{GPR} = 0.528$
True Duffing oscillator with F=0.357, $\ddot{x} = x - 0.5\dot{x} - x^3 + 0.357cos(t)$			
With Gaussian processes	$\ddot{x} = 0.995x - 0.500\dot{x} - 0.997x^3 + 0.359cos(t)$, $R^2_{True} = 0.99998$, $R^2_{GPR} = 0.9995$	$\ddot{x} = 0.986x - 0.499\dot{x} - 0.991x^3 + 0.361cos(t)$, $R^2_{True} = 0.99987$, $R^2_{GPR} = 0.996$	$\ddot{x} = 0.950x - 0.486\dot{x} - 0.968x^3 + 0.365cos(t)$, $R^2_{True} = 0.998$, $R^2_{GPR} = 0.966$
Without Gaussian processes	$\ddot{x} = 0.989x - 0.493\dot{x} - 0.988x^3 + 0.352cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.991$	$\ddot{x} = 0.916x - 0.436\dot{x} - 0.913x^3 + 0.321cos(t)$, $R^2_{True} = 0.992$, $R^2_{GPR} = 0.918$	$\ddot{x} = -0.339x^2 + 0.386cos(x)$, $R^2_{True} = 0.739$, $R^2_{GPR} = 0.522$
True Duffing oscillator with F=0.358, $\ddot{x} = x - 0.5\dot{x} - x^3 + 0.358cos(t)$			
With Gaussian processes	$\ddot{x} = 0.995x - 0.500\dot{x} - 0.997x^3 + 0.359cos(t)$, $R^2_{True} = 0.999986$, $R^2_{GPR} = 0.9996$	$\ddot{x} = 0.984x - 0.497\dot{x} - 0.989x^3 + 0.361cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.997$	$\ddot{x} = 0.945x - 0.481\dot{x} - 0.963x^3 + 0.364cos(t)$, $R^2_{True} = 0.998$, $R^2_{GPR} = 0.970$
Without Gaussian processes	$\ddot{x} = 0.989x - 0.493\dot{x} - 0.988x^3 + 0.354cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.991$	$\ddot{x} = 0.917x - 0.438\dot{x} - 0.913x^3 + 0.324cos(t)$, $R^2_{True} = 0.992$, $R^2_{GPR} = 0.917$	$\ddot{x} = -0.340x^2 + 0.388cos(x)$, $R^2_{True} = 0.738$, $R^2_{GPR} = 0.519$
True Duffing oscillator with F=0.4, $\ddot{x} = x - 0.5\dot{x} - x^3 + 0.4cos(t)$			
With Gaussian processes	$\ddot{x} = 0.995x - 0.499\dot{x} - 1.001x^3 + 0.399cos(t)$, $R^2_{True} = 0.99989$, $R^2_{GPR} = 0.998$	$\ddot{x} = 0.978x - 0.493\dot{x} - 0.995x^3 + 0.395cos(t)$, $R^2_{True} = 0.9993$, $R^2_{GPR} = 0.983$	$\ddot{x} = 0.889x + -0.450\dot{x} + -0.949x^3 + 0.364cos(1t)$, $R^2_{True} = 0.991$, $R^2_{GPR} = 0.863$
Without Gaussian processes	$\ddot{x} = 0.961x - 0.477\dot{x} - 0.961x^3 + 0.383cos(t)$, $R^2_{True} = 0.996$, $R^2_{GPR} = 0.957$	$\ddot{x} = 0.657x - 0.259\dot{x} - 0.722x^3 -$ $0.095\ x^2\dot{x} + 0.267x\dot{x}^2 + 0.266cos(t)$, $R^2_{True} = 0.933$, $R^2_{GPR} = 0.748$	$\ddot{x} = -0.142x^3 + 0.326x\dot{x}^2$, $R^2_{True} = 0.349$, $R^2_{GPR} = 0.230$

are added, the computation time rises with $O(n^3)$, which becomes costly. A more computationally efficient solution is to train multiple Gaussian processes on different time-spans and then combine the data before the symbolic regression step. For example, if a timespan between 2000s and 3000s is needed, one can train a Gaussian process between 2000s and 2100s, 2500s-2600s and 2900s-3000s, with 2000 datapoints for each interval. The trained Gaussian processes can then predict more datapoints within their respective intervals. This can be inputted as three trajectories in the SINDy algorithm. This way governing equations can be more robustly discovered from chaotic systems.

D. Generality of the strategy

The strategy of first using a noise reduction method and then a symbolic regressor is not limited to Gaussian process regression and SINDy. In order to demonstrate the generality of the strategy, the combination of splines and AI Feynman was also tested. Spline regression is a regression method that, like Gaussian processes, smoothen data. In a similar way, splines can be used to reduce the noise in data by predicting new smoothened data. The predicted data is then used as input for AI Feynman, another symbolic regressor. The effect of combining splines and AI Feynman is consistent with the results for the combination of Gaussian processes and SINDy. It was found that by adding a noise reduction step with splines, equations could be found with higher amounts of added noise than with AI Feynman alone.

VIII. Discussion

A common critique of symbolic regressors is the discrepancy between results on clean synthetic data and experimental data. In this work, an effort has been made to reduce this discrepancy by looking at the challenges in

TABLE II: Results for all test problems without derivative information, with different amounts of noise. Percentages are relative to the magnitude of the clean signal. R^2_{True} is the R^2 compared to the true system and R^2_{GPR} is the R^2 compared to the smoothened system.

Functions	**True equation**	**5% noise added**	**10% noise added**
Van der pol oscillator	$\ddot{x} = -x + 2.5\dot{x} - 2.5^2\dot{x}$	$\ddot{x} = -0.997x + 2.202\dot{x} - 2.158x^2\dot{x}$, $R^2_{True} = 0.984$, $R^2_{GPR} = 0.919$	$\ddot{x} = -1.004x + 1.896\dot{x} - 1.880^2\dot{x}$, $R^2_{True} = 0.948$, $R^2_{GPR} = 0.839$
Lorenz attractor	$\dot{x} = 10y - 10x$ $\dot{y} = 28x - y - xz$ $\dot{z} = xy - 2.667z$	$\dot{x} = 9.754y - 9.805x$ $\dot{y} = 24.721x - 0.887xz - 0.758y$, $\dot{z} = 0.912xy - 2.433z$ $R^2_{True} = 0.979$, $R^2_{GPR} = 0.990$	$\dot{x} = 9.956y - 10.030x$ $\dot{y} = 22.080x - 0.820xz$, $\dot{z} = 0.917xy - 2.569z + 2.575$ $R^2_{True} = 0.990$, $R^2_{GPR} = 0.958$
Duffing oscillator F=0.325, period 1	$\ddot{x} = x - 0.5\dot{x} - x^3 + 0.325cos(t)$	$\ddot{x} = 0.998x - 0.477\dot{x} - 0.998x^3 + 0.320cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.992$	$\ddot{x} = 0.993x - 0.552\dot{x} - 1.003x^3 + 0.348cos(t)$, $R^2_{True} = 0.9992$, $R^2_{GPR} = 0.981$
Duffing oscillator F=0.348, period 2	$\ddot{x} = x - 0.5\dot{x} - x^3 + 0.348cos(t)$	$\ddot{x} = 0.994x - 0.490\dot{x} - 0.996x^3 + 0.341cos(t)$, $R^2_{True} = 0.99996$, $R^2_{GPR} = 0.994$	$\ddot{x} = 0.985x - 0.495\dot{x} - 0.989x^3 + 0.340cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.980$
Duffing oscillator F=0.357, period 4	$\ddot{x} = x - 0.5\dot{x} - x^3 + 0.357cos(t)$	$\ddot{x} = 0.999x - 0.495\dot{x} - 1.001x^3 + 0.352cos(t)$, $R^2_{True} = 0.99998$, $R^2_{GPR} = 0.995$	$\ddot{x} = 0.985x - 0.500\dot{x} - 0.990x^3 + 0.348cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.979$
Duffing oscillator F=0.358, period 8	$\ddot{x} = x - 0.5\dot{x} - x^3 + 0.358cos(t)$	$\ddot{x} = 0.995x - 0.509\dot{x} - 0.996x^3 + 0.364cos(t)$, $R^2_{True} = 0.99996$, $R^2_{GPR} = 0.995$	$\ddot{x} = 0.989x - 0.525\dot{x} - 0.993x^3 + 0.373cos(t)$, $R^2_{True} = 0.9998$, $R^2_{GPR} = 0.983$
Duffing oscillator F=0.4, chaos, True	$\ddot{x} = x - 0.5\dot{x} - x^3 + 0.4cos(t)$	$\ddot{x} = 0.986x - 0.497\dot{x} - 0.990x^3 + 0.394cos(t)$, $R^2_{True} = 0.9990$, $R^2_{GPR} = 0.983$	$\ddot{x} = 0.967x - 0.529\dot{x} - 0.987x^3 + 0.397cos(t)$, $R^2_{True} = 0.998$, $R^2_{GPR} = 0.971$

experimental data. With the proposed method, more noise can be handled, the points do not need to be equally distributed, and more points can be predicted if too few points are measured. The proposed hyperparameter optimisation makes it possible to choose an equation from many possible ones that SINDy provides.

We have demonstrated that for numerically calculating the derivatives, Gaussian processes significantly improve the accuracy. For other numerical differentiators, the score improves drastically if the data provided is predicted by a Gaussian process. This way derivatives can be more robustly calculated and this thus makes the overall process much more robust to noise.

The discovered equations can be used to gain further understanding of the researched system and can be used to build descriptive models. In this work, the examples are taken from the field of dynamics but the algorithm can be applied to many more fields where models are needed, such as ecology, finance, geoscience, engineering, chemistry, biology and physics.

In case of failure, the algorithm often still leaves us with useful information about the true equation. Even though the form does not exactly match the form of the true equation any more. It is often still similar, where either small extra terms are added to overfit to the noise or the equation underfits by leaving out one of the correct terms.

The algorithm works well for white noise but the approach is still lacking for other types of noise. A problem with coloured noise is that Gausian processes have a hard time differentiating between trends in the signal and trends in the noise. In practise the regressed signal will include the trend from the noise, which causes the algorithm to find wrong equations. Furthermore the algorithm is likely to underfit if the required equation is long. This happens because longer equations are punished due to regularisation. Another problem with longer equations are the complex terms. The algorithm will work much better if the researcher has an idea of what terms make up the equation. If the search space is too large, the results will become worse. If the equation has complex terms, the researcher would need to know this beforehand. Otherwise, the equation cannot be found.

Machine learning techniques provide novel approaches towards research. Together with the availability of bigger datasets, we believe these techniques will only become more relevant in the years to come. Although these techniques require less knowledge of the physics to find new equations, we believe that these techniques, if used correctly, will only facilitate researchers with grasping and tackling more complex problems than they would without these techniques.

REFERENCES

[1] L. Ljung, "System identification," in *Signal Analysis and Prediction*. Birkhäuser Boston, 1998, pp. 163–173. [Online]. Available: https://doi.org/10.1007/978-1-4612-1768-8_11

[2] M. Mostowfi, M. Matin, M. Aslanzad, A. F. Azmayesh, L. Vakilian, E. Sofroni, M. Islam, and R. Liu, "Model-building cartesian genetic programming methodology," 2013.

[3] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the*

National Academy of Sciences, vol. 113, no. 15, pp. 3932–3937, 2016. [Online]. Available: https://www.pnas.org/content/113/15/3932

[4] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009. [Online]. Available: https://science.sciencemag.org/content/324/5923/81

[5] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 104, no. 24, pp. 9943–9948, 2007. [Online]. Available: https://www.pnas.org/content/104/24/9943

[6] S.-M. Udrescu and M. Tegmark, "Ai feynman: A physics-inspired method for symbolic regression," *Science Advances*, vol. 6, no. 16, 2020. [Online]. Available: https://advances.sciencemag.org/content/6/16/eaay2631

[7] B. K. Petersen, "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," 2020.

[8] S. CHEN, S. A. BILLINGS, C. F. N. COWAN, and P. M. GRANT, "Non-linear systems identification using radial basis functions," *International Journal of Systems Science*, vol. 21, no. 12, pp. 2513–2539, 1990. [Online]. Available: https://doi.org/10.1080/00207729008910567

[9] I. J. LEONTARITIS and S. A. BILLINGS, "Input-output parametric models for non-linear systems part i: deterministic non-linear systems," *International Journal of Control*, vol. 41, no. 2, pp. 303–328, 1985. [Online]. Available: https://doi.org/10.1080/0020718508961129

[10] ——, "Input-output parametric models for non-linear systems part ii: stochastic non-linear systems," *International Journal of Control*, vol. 41, no. 2, pp. 329–344, 1985. [Online]. Available: https://doi.org/10.1080/0020718508961130

[11] S. Billings, "Nonlinear system identification: Narmax methods in the time, frequency, and spatio-temporal domains," *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*, 08 2013.

[12] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep neural networks for data-driven discovery of nonlinear dynamical systems," 2018.

[13] A. Cochocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, 1st ed. USA: John Wiley amp; Sons, Inc., 1993.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] J. Kocijan, A. Girard, B. Banko, and R. Murray-Smith, "Dynamic systems identification with gaussian processes," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 11, no. 4, pp. 411–424, 2005. [Online]. Available: https://doi.org/10.1080/13873950500068567

[16] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Machine learning of linear differential equations using gaussian processes," *Journal of Computational Physics*, vol. 348, p. 683–693, Nov 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2017.07.050

[17] M. Raissi and G. E. Karniadakis, "Hidden physics models: Machine learning of nonlinear partial differential equations," *Journal of Computational Physics*, vol. 357, p. 125–141, Mar 2018. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2017.11.039

[18] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Inferring solutions of differential equations using noisy multi-fidelity data," *Journal of Computational Physics*, vol. 335, pp. 736 – 746, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999117300761

[19] M. Raissi, P. Perdikaris, and G. Karniadakis, "Numerical gaussian processes for time-dependent and nonlinear partial differential equations," *SIAM Journal on Scientific Computing*, vol. 40, 03 2017.

[20] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton, "Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data," *Journal of Open Source Software*, vol. 5, no. 49, p. 2104, 2020. [Online]. Available: https://doi.org/10.21105/joss.02104

[21] H. Haken, "Analogy between higher instabilities in fluids and lasers," *Physics Letters A*, vol. 53, no. 1, pp. 77 – 78, 1975. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0375960175903539

[22] E. Knobloch, "Chaos in the segmented disc dynamo," *Physics Letters A*, vol. 82, no. 9, pp. 439 – 440, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0375960181902747

[23] K. M. Cuomo and A. V. Oppenheim, "Circuit implementation of synchronized chaos with applications to communications," *Phys. Rev. Lett.*, vol. 71, pp. 65–68, Jul 1993. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.71.65

[24] D. Poland, "Cooperative catalysis and chemical chaos: a chemical model for the lorenz equations," *Physica D: Nonlinear Phenomena*, vol. 65, no. 1, pp. 86 – 99, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016727899390006M

III

Closing

5

Conclusions and recommendations

The objective of this work is to make symbolic regression more noise-robust, to robustly numerically calculate the derivatives and to choose the best equation from a list of equations provided by a symbolic regressor. To this end, different regressors for noise reduction, symbolic regressors and machine learning methods have been employed. This section reports the main conclusions of these methods. Furthermore, recommendations for further research are given.

5.1. Conclusions

The general strategy to first reduce the noise, makes symbolic regression more robust to noise. Multiple methods have been tried with the aim of noise reduction. It was concluded that Gaussian process regression was the most robust to high amounts of added noise and that there are additional benefits, such as being able to provide uncertainty bounds and work with irregularly sampled, scarce and sparse data. Other methods such as neural networks and splines can also be used as noise-reduction methods but were found to be less effective.

Gaussian processes increase the noise-robustness for numerically calculating the derivatives. For other numerical differentiators, the accuracy also improves if the data provided is predicted by a Gaussian process. This way it becomes possible to discover differential equations, even if only positional data is available.

The strategy is less effective against coloured noise. The described regression methods have trouble differentiating trends in the signal from trends in the noise. This makes the noise-reduction step less beneficial than for white noise.

For the symbolic regression SINDy provided the most consistent results and has a small noise tolerance on its own. As genetic programming relies on randomised

candidate functions, it was less consistent. AI Feynman's noise robustness is heavily dependent on the function to be discovered, which makes it less useful for comparison. Overall SINDy has the highest noise tolerance but SINDy also needs more understanding of the system than the other symbolic regressors, which have a truly independent structure search.

It was concluded that by skipping data between the training and validation data, together with L1-regularisation, it is possible to choose an equation from a list of possible equations found by a symbolic regressor.

5.2. Recommendations

It is recommended to use SymbolicAI on experimental data. This way new equations can be discovered from data. In order to improve SymbolicAI the following recommendations are presented:

Further research is needed to see if a neural network can be constructed that has better noise-reduction properties than Gaussian processes. Bayesian neural networks allow for uncertainty estimation and averaging over samples could have a similar smoothing effect as Gaussian processes. Since Neural networks can more easily handle more datapoints than Gaussian processes it could reduce the noise more than Gaussian processes for cases where large amounts of data are available. Special care has to be taken to prevent overfitting. For this, different types of regularisation methods can be examined.

Neural networks and Gaussian processes can also be combined in deep Gaussian processes. This is a neural network, where each neuron is a Gaussian process. Limits in computing power restrict the size of deep Gaussian processes and make it a slow process. However, deep Gaussian processes may provide better results for noise reduction than Gaussian processes or Neural networks alone. Especially, with the increasing strength of computers, this technique may become much more relevant.

The uncertainty bounds provided by Gaussian processes can also be passed on to the symbolic regressor to provide uncertainty estimates of the found equation. This can be done by testing how many standard deviations away from the mean the same equation can be discovered.

In SINDy, the library of candidate functions is user-defined. In order to make the algorithm less dependent on the understanding of the systems, a search of the candidate functions can be designed. One implementation could be to find the terms using genetic programming. For example, if genetic programming finds terms like $0.5x^3$, $cos(x)$ and $2x^2$, all polynomial terms up to order 3 and trigonometric terms can make up the SINDy library.

The approach for AI Feynman is to identify simplifying properties in a dataset and to exploit these properties to create simplified sub-problems that can be solved using simple techniques like a polynomial fit. A combination of AI Feynman's simplification process and stronger tools, such as SINDy, to solve the sub-problems might

increase the strength of both methods and could allow the researcher to find more complicated equation than with SINDy alone.

Coloured noise
To reduce coloured noise, one can create multiple trajectories on the same timespan and use SymbolicAI to find an equation for each trajectory. These equations can be compared and the shared terms of the equation are kept. It is postulated that the extra non-shared terms describe the trend from the coloured noise. However, this assumes modularity of terms that describe the trend from the signal and terms that describe the trend from the noise. Therefore extra care should be taken to prevent underfitting, where a smaller amount of terms are found that simultaneously describe the noise and the signal and are thus not modular.

One could also use Gaussian processes for each trajectory and then take the average of all these Gaussian processes. This should reduce the extra trend from the coloured noise. The equations can then be discovered with the averaged predicted data.

Another possibility is to try to learn the trend of noise directly through Gaussian process regression. Here a kernel can be constructed that learns the signal separately from the noise, in a similar way as a white noise kernel learns white noise.

One can also look at the frequency domain and apply a filter. This, however, requires knowledge of the equation and the researcher should be able to identify the signal from a plot of the Fourier transform of the data.

Another possibility is to modulate the amplitude of the data in the frequency domain to be more constant. One can do this by first finding the trend in the Fourier transform of the data (e.g. $1/f$) and then modulating the amplitude to a constant trend. This way the noise will become closer to white noise and the signal becomes scaled. The noise can then be reduced with Gaussian process regression and the signal can be unscaled.

It is also possible to force a repeating pattern if the researcher knows there should be a certain periodicity in the data. This way the extra trend caused by coloured noise can be identified.

References

[1] I. J. LEONTARITIS and S. A. BILLINGS, *Input-output parametric models for non-linear systems part i: deterministic non-linear systems,* International Journal of Control **41**, 303 (1985), https://doi.org/10.1080/0020718508961129 .

[2] I. J. LEONTARITIS and S. A. BILLINGS, *Input-output parametric models for non-linear systems part ii: stochastic non-linear systems,* International Journal of Control **41**, 329 (1985), https://doi.org/10.1080/0020718508961130 .

[3] S. Billings, *Nonlinear system identification: Narmax methods in the time, frequency, and spatio-temporal domains,* Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains (2013), 10.1002/9781118535561.

[4] G. Cybenko, *Approximation by superpositions of a sigmoidal function,* Mathematics of Control, Signals and Systems **2**, 303 (1989).

[5] G. E. Hinton and R. M. Neal, *Bayesian learning for neural networks,* (1995).

[6] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)* (The MIT Press, 2005).

[7] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems,* Proceedings of the National Academy of Sciences **113**, 3932 (2016), https://www.pnas.org/content/113/15/3932.full.pdf .

[8] M. Schmidt and H. Lipson, *Distilling free-form natural laws from experimental data,* Science **324**, 81 (2009), https://science.sciencemag.org/content/324/5923/81.full.pdf .

[9] J. Bongard and H. Lipson, *Automated reverse engineering of nonlinear dynamical systems,* Proceedings of the National Academy of Sciences **104**, 9943 (2007), https://www.pnas.org/content/104/24/9943.full.pdf .

[10] R. Tibshirani, *Regression shrinkage and selection via the lasso,* JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B **58**, 267 (1994).

[11] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Multistep neural networks for data-driven discovery of nonlinear dynamical systems,* (2018), arXiv:1801.01236 [math.DS] .

[12] J. Kanter and K. Veeramachaneni, *Deep feature synthesis: Towards automating data science endeavors,* (2015) pp. 1–10.

[13] A. Hubin, G. Storvik, and F. Frommlet, *Deep bayesian regression models.* (2018).

[14] A. Wuraola and N. Patel, *Computationally efficient radial basis function,* in *Neural Information Processing*, edited by L. Cheng, A. C. S. Leung, and S. Ozawa (Springer International Publishing, Cham, 2018) pp. 103–112.

[15] M. Mongillo, *Choosing basis functions and shape parameters for radial basis function methods,* SIAM Undergraduate Research Online **4** (2011), 10.1137/11S010840.

[16] A. K. Alexandridis and A. D. Zapranis, *Wavelet neural networks: A practical guide,* Neural Networks **42**, 1 (2013).

[17] Z. Zainuddin and P. Ong, *Function approximation using artificial neural networks,* WSEAS Transactions on Mathematics **7** (2008).

[18] E. Gilboa, Y. Saatci, and J. P. Cunningham, *Scaling multidimensional inference for structured gaussian processes,* IEEE Transactions on Pattern Analysis and Machine Intelligence **37**, 424–436 (2015).

[19] A. J. Smola and P. L. Bartlett, *Sparse greedy gaussian process regression,* in *Advances in Neural Information Processing Systems 13*, edited by T. K. Leen, T. G. Dietterich, and V. Tresp (MIT Press, 2001) pp. 619–625.

[20] C. K. I. Williams and M. Seeger, *Using the nyström method to speed up kernel machines,* in *Advances in Neural Information Processing Systems 13*, edited by T. K. Leen, T. G. Dietterich, and V. Tresp (MIT Press, 2001) pp. 682–688.

[21] L. Csató and M. Opper, *Sparse on-line gaussian processes,* Neural Computation **14**, 641 (2002), https://doi.org/10.1162/089976602317250933 .

[22] M. McIntire, D. Ratner, and S. Ermon, *Sparse gaussian processes for bayesian optimization,* in *UAI* (2016).

[23] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Data-driven discovery of partial differential equations,* Science Advances **3** (2017), 10.1126/sciadv.1602614, https://advances.sciencemag.org/content/3/4/e1602614.full.pdf .

[24] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Inferring biological networks by sparse identification of nonlinear dynamics,* IEEE Transactions on Molecular, Biological and Multi-Scale Communications **2**, 52–63 (2016).

[25] S.-M. Udrescu and M. Tegmark, *Ai feynman: A physics-inspired method for symbolic regression,* Science Advances **6** (2020), 10.1126/sciadv.aay2631, https://advances.sciencemag.org/content/6/16/eaay2631.full.pdf .

[26] T. Stephens, *gplearn's documentation,* (2019).

[27] A. BelÃ, C. Pascual, D. MÃ, T. BelÃ, and C. Neipp, *Exact solution for the nonlinear pendulum,* Revista Brasileira de Ensino de FÃsica **29**, 645 (2007).

[28] D. Duvenaud, *Automatic model construction with gaussian processes,* (2014), 10.17863/CAM.14087.

[29] A. McHutchon, *Differentiating gaussian processes,* (2013).

IV

appendices

A

Appendix literature review

A

A.1. Appendix - The need for the derivatives

One limitation for current methods of symbolic regression is that for a differential equation you would need the positions and its derivatives. By numerically calculating the derivatives, the error is increased so it would make sense to try to eliminate the need for the derivatives. In [11] is a proposal to use neural networks to create a fitting over data. In this paper, there is no need to know the second-order derivatives in a second-order differential equation and no need to know the first-order derivatives in a first-order differential equation. This is because use is made of multi-step time-stepping schemes. If we try to implement this in symbolic regression as well, an example algorithm can be created, like shown below. Please note that this algorithm is specifically for a second order differential equation. This algorithm is more robust, as numerical integration is more robust than numerical differentiation.

1. Collect experimental data from experiments (e.g. pendulum time series) or from other sources
2. Either collect the first derivative or numerically calculate the first derivative
3. Divide the data into training data and test data (this time the data does not need to be labelled)
4. Estimate the formula
   (a) Try a random formula
   (b) Numerically integrate the found output with multi-step time-stepping schemes
   (c) Compare the result from the previous step to the positions in a cost function (e.g. mean absolute error)
   (d) If convergence criteria are met stop here, otherwise try again with a new formula that is adapted from earlier iterations.
5. Use test data for validation and estimating the accuracy
6. Numerically integrate output to obtain new data

It is not possible to eliminate the need for the first-order derivatives in a second-order differential equation. This is shown in Appendix A.2.

A.2. Appendix - Derivative investigation

Here, there will be looked at the possibility to eliminate the need for the first-order derivatives in a second-order differential equation. In order to examine this, the simple pendulum is taken as an example. The equation for the second-order derivative A.1 is dependent on both the first-order derivative and the position.

$$\ddot{\theta} = -g/L \sin(\theta) - \alpha\dot{\theta} \tag{A.1}$$

Since the exact solution requires both, the first-order derivative is part of the input. It can therefore not be excluded from the algorithm. Instead, the idea is to find an additional formula relating the first-order derivative to the position, $\dot{\theta} = f(\theta, t, \theta_0, \omega_0)$. This can be done analytically by considering the total energy in the system. This results in equation A.2:

$$\dot{\theta} = \sqrt{\frac{2mg(L - L * cos(\theta_0)) + mL^2\dot{\theta_0}^2 - 2mg(L - Lcos(\theta)) - 2h(t)}{mL^2}} \tag{A.2}$$

where $h(t)$ is the energy loss due to friction:

$$h(t) = -mgh - \frac{1}{2}mv^2 + E_0 \tag{A.3}$$

In an effort to examine if the energy loss can be approximated the loss is plotted as shown in figure A.1. From the figure, it can be concluded that the formula needed would be very complicated. The assumption was made that trying to find this formula with symbolic regression would induce more error than numerically calculating the derivatives.

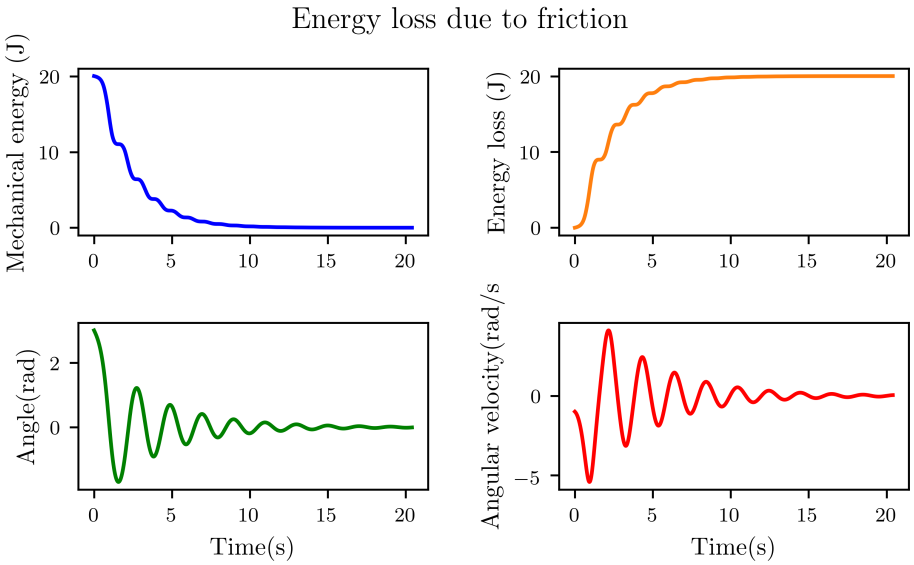


Figure A.1: Energy loss due to friction in a simple pendulum

One could also try to find the solution of the differential equation and try to find $\theta(t, \theta_0, \theta_0)$ directly. In Beléndez et al., 2007 [27] this was done for a simple pen-

dulum without friction. The equation found was:

$$\theta(t) = 2\arcsin\left\{\sin\frac{\theta_0}{2} sn\left[K\left(\sin^2\frac{\theta_0}{2}\right) - \omega_0 t; \sin^2\frac{\theta_0}{2}\right]\right\} \tag{A.4}$$

with sn(u;m) as the Jacobi elliptic function and:

$$K(m) = \int_0^1 \frac{dz}{\sqrt{(1-z^2)(1-mz^2)}} \tag{A.5}$$

Here again the complexity of the formulas would make it more advantageous to numerically calculate the derivatives. It can, therefore, be concluded that it is more advantageous to either measure or numerically calculate the first-order derivatives in a general algorithm for finding a second-order differential equation.

B

Preprocessing

B.1. Appendix - Gaussian processes regression

Unlike many regression models, Gaussian process regression does not attempt to learn every parameter in a function. Instead it creates a distribution over a function where it takes a Bayesian approach that is non-parametric. Using probabilistic inference has many advantages over non-probabilistic methods. It is an effective approach to prevent overfitting and the samples give an insight in the possible structures in the data.

With Gaussian processes many samples are taken from a high-dimensional multivariate Gaussian distribution. The covariance of the distribution is conditioned through the ”closeness” of the provided datapoints. The closeness is determined through kernel functions that take the distance between two points to determine the correlation. Subsequently, the average of the many samples is taken, which acts as the prediction line, with a confidence interval. An example of the result is shown in Figure B.1. On the left, it can be seen that the confidence interval is 0 on training points with no noise and quite large where there are no training points. As can be seen on the right, increasing the number of training points result in a far more accurate fitting. A Gaussian process is defined as follows:

$$f(\mathrm{x}) \sim \mathcal{GP}(m(\mathrm{x}), K(\mathrm{x}, \mathrm{x}')) \tag{B.1}$$

An example of a kernel function is shown in equation B.2, where each element in kernel matrix K is defined as shown.

$$k(x_{,}x_j) = \sigma_f^2 e^{\frac{1}{-2l^2}(x_i - x_j)^2} \tag{B.2}$$

Since Gaussian process processes provide a distribution over functions we can take a function as a sample. in figure B.2 an example can be seen where we draw 10

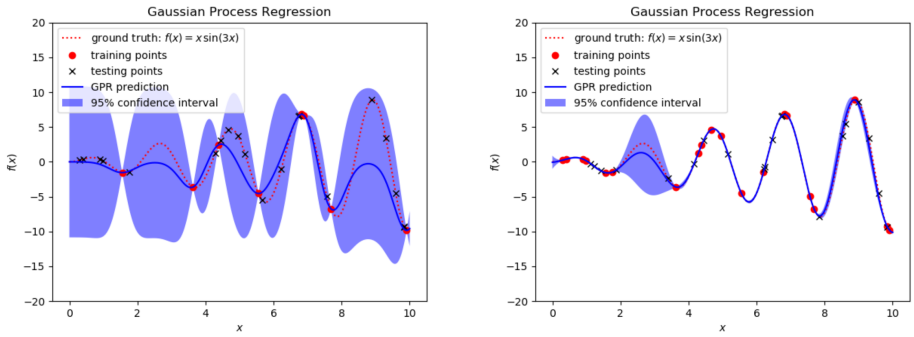


Figure B.1: An example of Gaussian Processes, Reprinted from imechanica.org, by M.A. Bessa, 2020, https://imechanica.org/node/23957

samples from the distribution. In practice, the mean and the covariance of these samples are more interesting. The mean will provide the predicted values and the covariance gives an uncertainty estimate. For given data y (zero-mean), the mean and covariance are calculated at M unseen inputs as shown in equations B.3 and B.4 [18].

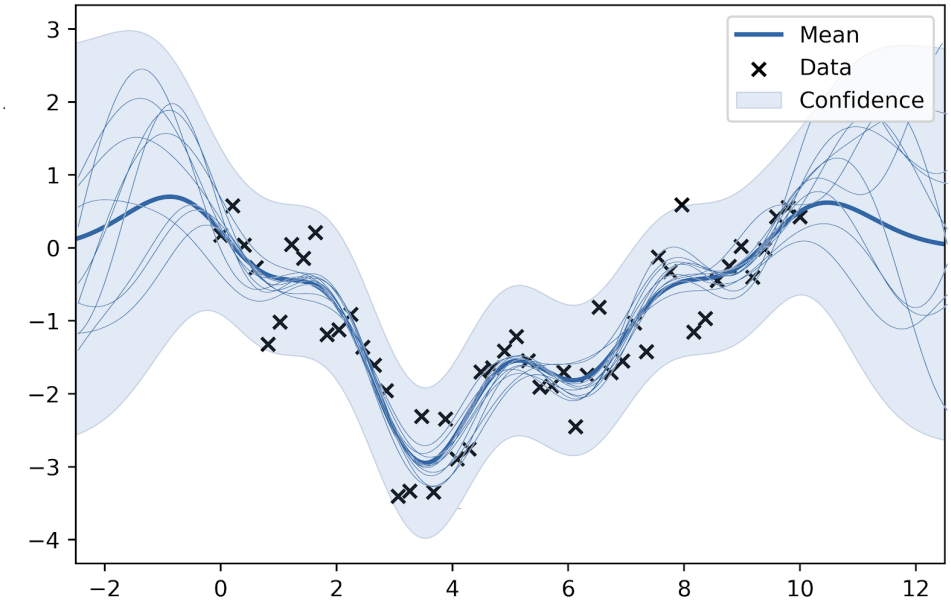


Figure B.2: Samples drawn from a Gaussian Process, as well as the confidence interval and mean

$$\mu_* = \mathrm{K}_{MN} \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{y} \tag{B.3}$$

$$\Sigma_\cdot = \mathrm{K}_M - \mathrm{K}_{A/N} \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{K}_{NM} \tag{B.4}$$

In equation B.2, σ_f and l are hyperparameters, which change the predictive quality and uncertainty. The best values for these hyperparameters, which are grouped in θ, are usually found through an optimisation process of the log marginal likelihood B.5 [6]. This has to be solved iteratively and is often done with Limited-memory BFGS (L-BFGS), a quasi newton optimization method. This process sometimes needs to be restarted with different starting points to find the global minimum, as sometimes it can get stuck in local minimum. An example with two local minima is shown in figure B.3.

$$\log Z(\theta) = -\frac{1}{2}\left[\mathrm{y}^\top \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{y} + N \log(2\pi) \; + \log\left|\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right|\right] \tag{B.5}$$

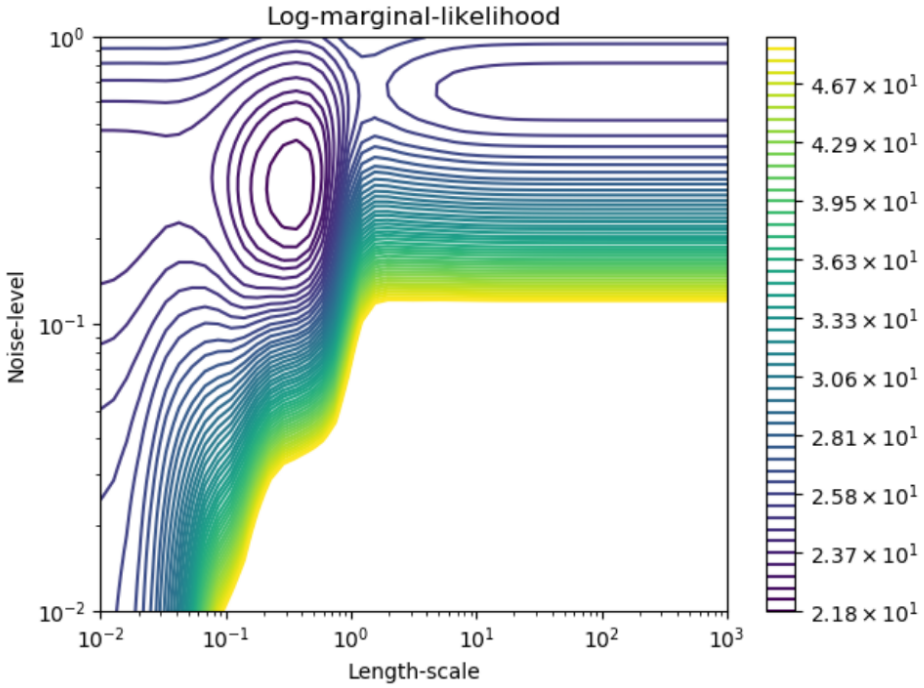


Figure B.3: Hyperparameters of kernel plotted against log marginal likelihood

B

B.1.1. Kernel engineering

The use of the correct kernel is central in Gaussian process regression. With the wrong kernel, the fit will be wrong as well. There exists no kernel that is correct for all problems. Through the use of kernel functions different datasets can be fitted in a different way. Kernel engineering is often seen as very intuitive and the effectiveness dependent on the experience of the engineer. However, there are certain rules of thumb. In figure B.4 different kernels are shown. In the graphs x represents the difference between two points and on the y-axis the corresponding value for the element of the covariance matrix $K(\mathbf{x}, \mathbf{x}'))$ is shown. The RBF (Squared-exponential) kernel is shown in equation B.2. It is important to note that many more kernels exist but these are very common.

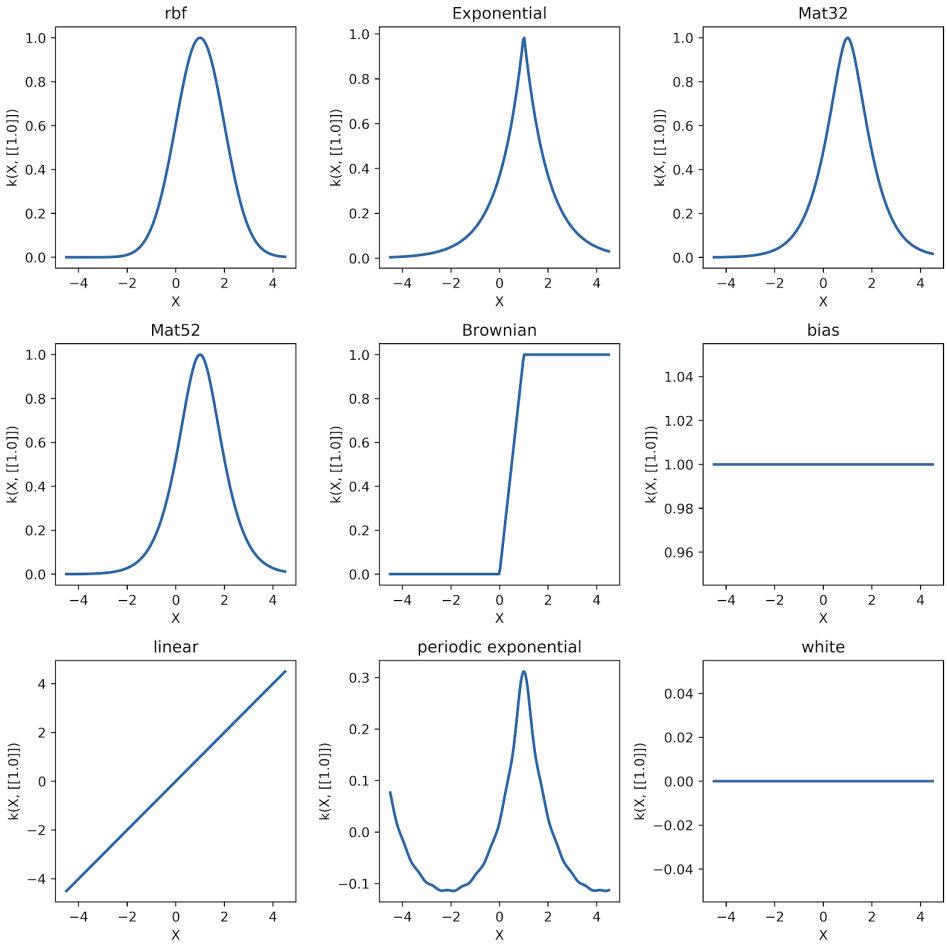


Figure B.4: Different kernel functions

In figure B.5 the use cases of four kernels are shown. These kernels are often used and provide good intuition. These kernels can also be combined through either multiplication or summation to achieve different effects, as shown in figure B.6.

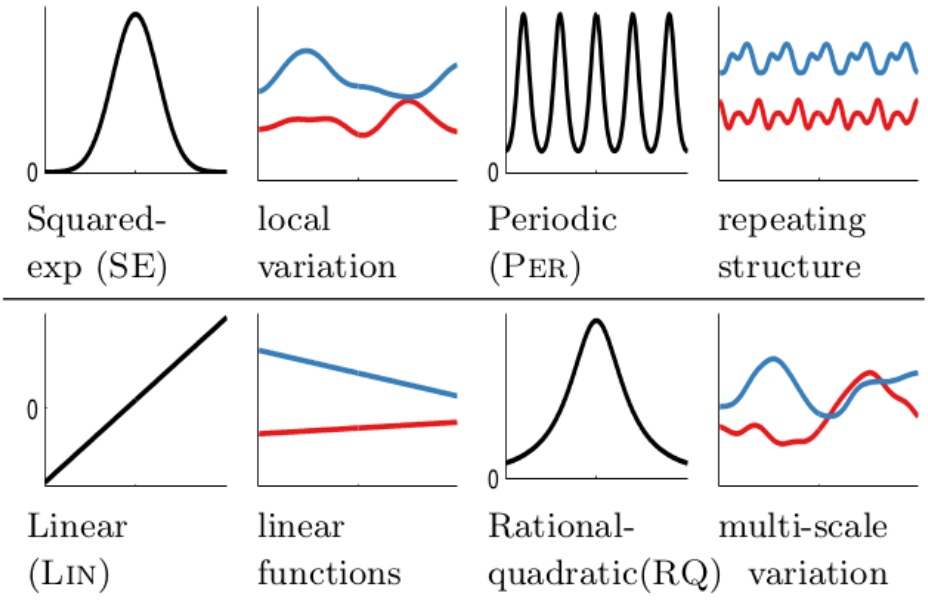


Figure B.5: Use case of different kernels, Reprinted from Duvenaud, David. (2014). Automatic model construction with Gaussian processes.

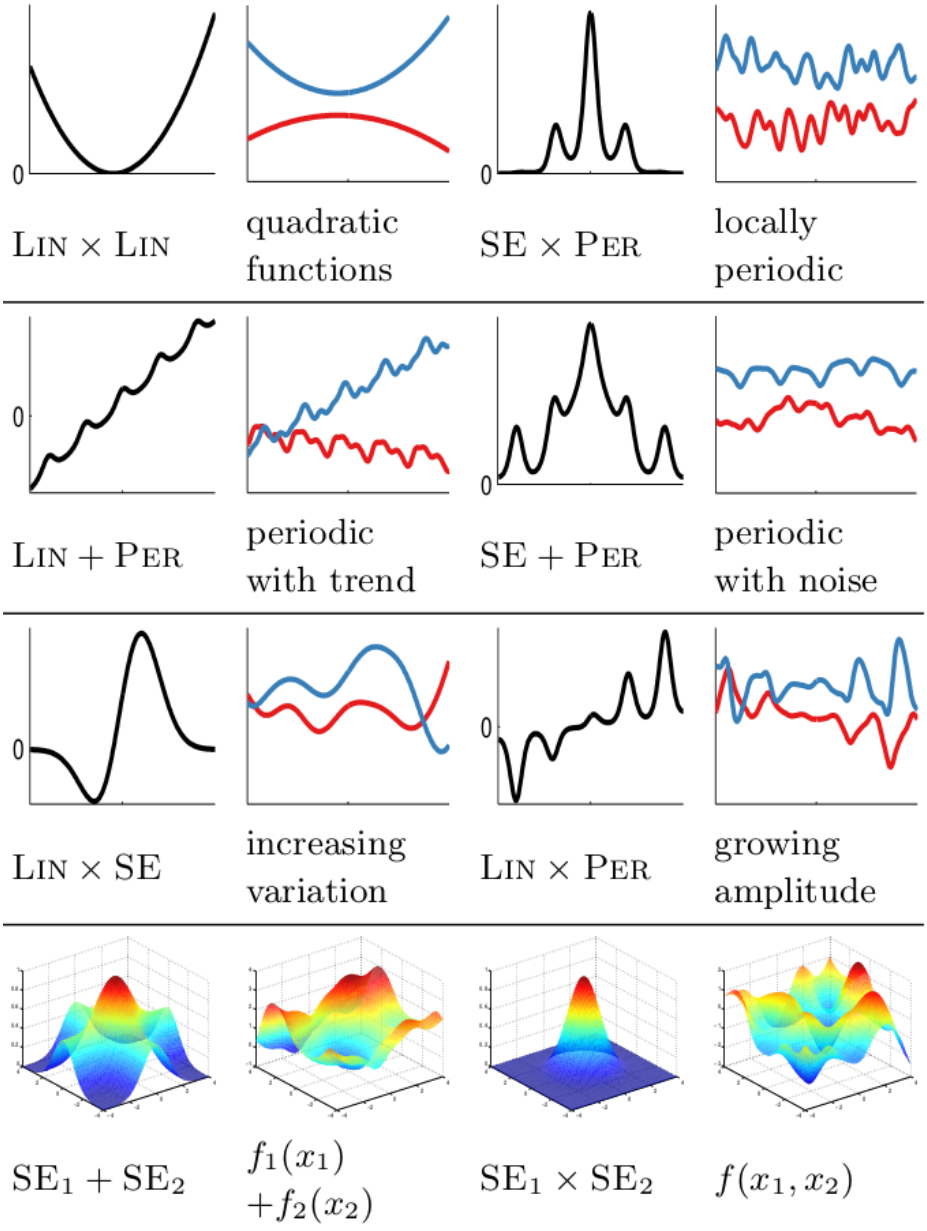


Figure B.6: Use case of combination of different kernels, Reprinted from Duvenaud, David. (2014). Automatic model construction with Gaussian processes.

As can be seen on the bottom row of image B.6, it is also possible to create combinations for higher dimensional regression. Through multiplication it is possible to

get similar properties in higher dimensions. This can be seen in figure B.7. It also possible to build an appropriate kernel for higher dimensional regression though summation, the effect of this is shown in figure B.8.

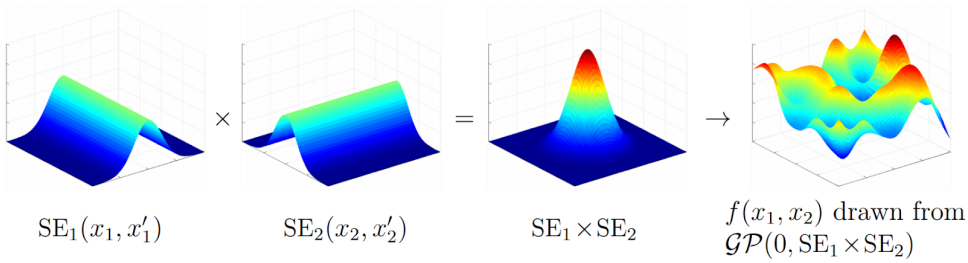


Figure B.7: Multiplication of kernels to achieve a kernel that is dependant on both dimensions, Reprinted from Duvenaud, David. (2014). Automatic model construction with Gaussian processes.

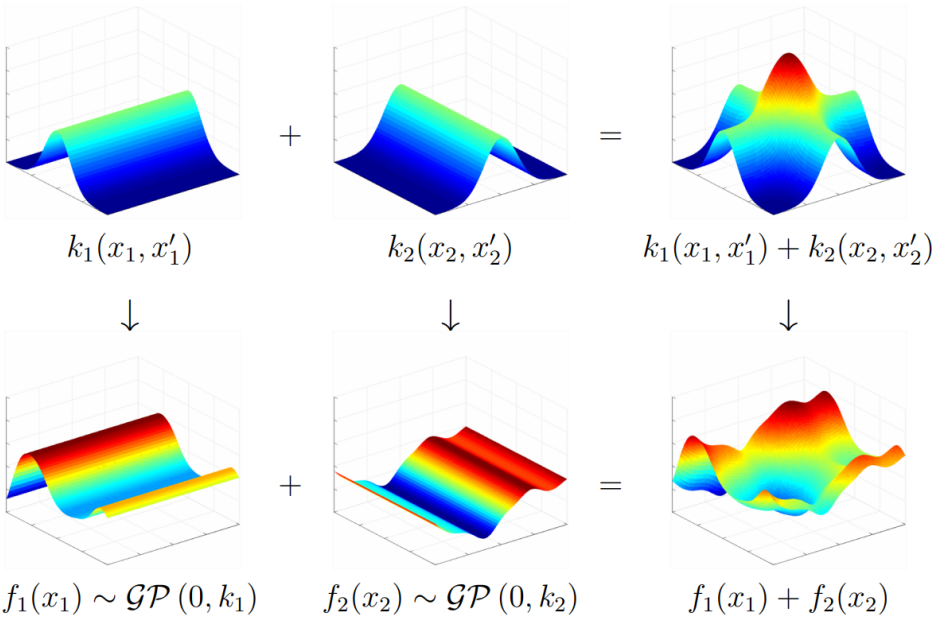


Figure B.8: Summation of kernels to achieve a kernel that is dependant on both dimensions, Reprinted from Duvenaud, David. (2014). Automatic model construction with Gaussian processes.

Many more combinations are possible and better combinations for a specific problem can be found through intuition or an automated kernel search, although an

automated kernel search has the downside of being computationally very costly. There are many more topics to explore in kernel engineering, such as changepoint kernels and expressing symmetries and invariances. For this and more information on kernel engineering one is encouraged to read ref [28].

B

B.1.2. Automated kernel search

The search for the right kernel can also be automated. This is helpful for the cases where one is not sure which kernel best fits the problem. In this case, a list of base kernels can be constructed from which an algorithm can find the best base kernel and construct, through multiplication and summation, an optimized kernel. The first step is to find the best fitting base kernel. The next step is to see if there exists a multiplication or summation with one of the base kernels that improve the fit. Then the algorithm checks if the first element can be replaced. By using the replacement step, the expression is allowed to evolve. This prevents the expression from being stuck in locally optimal solutions. For example, in a case where in the first iteration the data was fitted with an RBF kernel and after the 2nd iteration, a periodic kernel was added to explain some periodic oscillations. The found kernel is now as follows:

$$K = RBF + PER \tag{B.6}$$

It might turn out that if the periodic kernel explains the oscillations in your data, the global trend could be better explained with a linear kernel rather than a RBF kernel. With the replacement step, the found kernel is now:

$$K = LIN + PER \tag{B.7}$$

This process then continues until no further improvements are found.

B.2. Python Package

In order to provide a generalized framework a package was created, which contains various useful utilities. The aim is to generate noisy data from a variety of functions and denoise it using either Gaussian processes or Neural networks. The regression types will be evaluated and the best data can be chosen. The imported package contains two classes: In the cell below we import a file with two custom classes and one custom function: Here's a summary of the tools provided:

takeSamples class:

What: Returns an instance with data based on your provided function

Methods:

1. *__Init__(func,name,dims,order,timeParams=[0,0.01,1],params=None, feature_names=None,initialConditions=None,fromtIszero=False)*:
   sets parameters, calls sample method or sample2 method depending on the

input *func*. *func* should be either a python function or list with values. If no initialConditions are given and *func* is a python function, they will be randomized between 0 and 1.

2. *sample(self)*: A function that provides values for the position, velocity and acceleration vs time, given the function *func*. Values are obtained through integration. If no function parameters are given, it will use the default parameters from the provided function *func*. If no time parameters are given, it will integrate from 0 to 1 with time steps of 0.01.

3. *sample2(self)*: A function sets values for the position, velocity and acceleration vs time, given the values provided in *func*.

4. *dataframe(self)*: A function that puts the values in a pandas dataframe.

5. *addNoise(self,noise,X_data=None,y_data=None,X_train=None,y_train=None, X_test=None,y_test=None,type_noise='white',decay=20*:
A function that adds noise to the values. If the noise is white values from a normal distribution with mean 0 and variance *noise* will be added to the data. If the noise is non-white the noise will added will be drawn from a distribution with an exponential decay *decay* in the fourier domain.

6. *plotFunction(self,t=None,X=None,name=None,plot3d=True,saveImages=False)*:
A plot function of the values. Produces the following plots:

   (a) if *plot3d*=True: A 3d plot where useful. For example, a phase space plot

   (b) Positions vs time

   (c) if order=1: Velocities vs time

   (d) if order=2: Accelerations vs time Saves the images in a folder images if *saveImages*=True.

7. *split(self,testset_ratio=0.25,seed=1987)*: A function that splits the data into a training and test set.

8. *scaling(self)*: A function that scales the data to be 0 mean and with a variance of 1.

9. *inverse_scaling(self,pred=None,regressed=True,plotting=True, X=True,y=True,saveImages=False)*: A function that unscales the data based on the scaling from the scaling method. With *pred* it is possible to unscale predicted data *pred*.

10. *GPregression(self,t=None,x=None,k=None,inp_dim=1,restarts=4,var=1, name="Insert variable name here if x is not a string",printing=True, singleRegression=False,saveImages=False,plotting=True)*:
A function that performs gaussian process regression. The kernel can be specified with *k*. The default kernel is an RBF kernel.

11. *NNregression(self,grid_search=0,X_train=None,y_train=None,X_test=None, y_test=None,input_dimensions=1,neurons1=10,neurons2=10, neurons3=10, neurons4=10, activation='relu',optimizer='adam',printing=True)*:
A function that performs neural network regression. Performs a gridsearch if *grid_search*=1.

12. *score(self,y_test=None,y_test_pred=None,printing=True)*: A function that calculates the R2-score, mean absolute error and mean square error.

Automate function
Since the process up until regression is similar for different functions the process can be automated with this function. The automate function generates data on the basis of *func* and puts it in a pandas dataframe. Based on the arguments, the data will be also be scaled to be 0 mean and have a variance of 1, splitted into training and test data, plotted and saved to a folder images and have noise added to it.

Parameters:

1. *func*: Function you want to analyse. Accepts either a list of lists, where each nested list are values of a function parameter or a python function, where after the values will be generated through integration.

2. *dims*: Number of dimensions, should be an integer.

3. *order*: Order of differential equation, 0 if it isn't a differential equation.

4. *plotname=systems*: Name you want to appear on plots, should be a string.

5. *timeParams=[0,0.1,10]*: Time parameters you want your data to be between, list of start time, time step size and end time in that order

6. *testset_ratio=0.25*: Ratio of data that will be used for the test data

7. *noise=None* = Amount of noise to add to the data

8. *plotting=False*: Whether you want to plot the data

9. *params=None*: In case you don't want to use the default parameters of function *func*

10. *saveImages=False*: saves the images in a folder named images

11. *feature_names=None*: names of variables for plots

12. *type_noise='white'*: The type of noise to be added. If the noise is white values from a normal distribution with mean 0 and variance *noise* will be added to the data. If the noise is non-white the noise will added will be drawn from a distribution with an exponential decay *decay* in the fourier domain.

13. *decay=20*: Controls the decay of non white noise in the fourier domain

14. *initialConditions=None*: The initial conditions for integration. If no initialConditions are given and *func* is a python function, they will be randomized between 0 and 1.
15. *fromtIsZero=False*: Controls wether integration starts from time = 0
16. *scaling=True*: Controls if the data is scaled to be 0 mean with a variance of 1 before regression
17. *splitting=True*: Controls if the data is splitted in train and test data

FindKern class:

What: greedy search for the best kernel combination on the basis of specified base kernels

Relevant methods:

1. *__init__(self, k_base, X, Y)*: Initializes parameters
2. *initial_model(self, noise = 1e-6, noise_fixed = False, printing = True)*: Finds the best kernel of the specified base kernels
3. *greedy_brute_opt(self, iterations=3, plotting=True, printing_basic=True, Xnew=None,x=None, y=None, with_replacement=False, scatter_on=True)*: Performs a greedy search for the best combination of kernels.

Other methods in the FindKern class are called on internally in the the relevant methods and should not called on externally.

Relevant arguments:

1. *k_base*: A list of GPy kernels. These will be building blocks for the to be found kernel.
2. *X*: Variable for which to predict values, often time will be used here.
3. *Y*: Variables to predict.
4. *Iterations*: Determines the amount of restarts for the optimization of the hyperparameters.
5. *with_replacement*: Determines whether the greedy search will only extend the found kernel or if it should also check if found terms can be replaced with other kernels from the kernel base.
6. *noise*: The variance of the noise you think is present
7. *noise_fixed*: If you are certain the variance of the noise is a certain value it can fixed in the search.
8. *printing*: Prints intermediate results
9. *plotting*: Plots the prediction with the found kernel

C

Numerically robust differentiation

C.1. Differentiating Gaussian processes

It can be challenging to numerically differentiate noisy data. If no extra precautions are taken, the noise can get greatly amplified. The equation for the derivative is shown in equation C.1. Suppose a discretised version of this equation, as shown in equation C.2, is used, where Δt is the time-step. In a noisy situation, the signal f(x) can be decomposed to the superposition of the clean signal $g(x)$ and the noise $\eta(x)$. This is shown in figure C.1. The equation for the discrete differentiation then becomes equation C.3. It can be seen in figure C.1 that the noise has very sharp transitions compared to the clean signal. If the noisy signal is then differentiated, the error due to noise is then greatly amplified. This effect is shown in equation C.2.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{C.1}$$

$$f'(x) = \frac{f(x+\Delta t) - f(x)}{\Delta t} \tag{C.2}$$

$$f'(x) = \frac{g(x+\Delta t) - g(x)}{\Delta t} + \frac{\eta(x+\Delta t) - \eta(x)}{\Delta t} \tag{C.3}$$

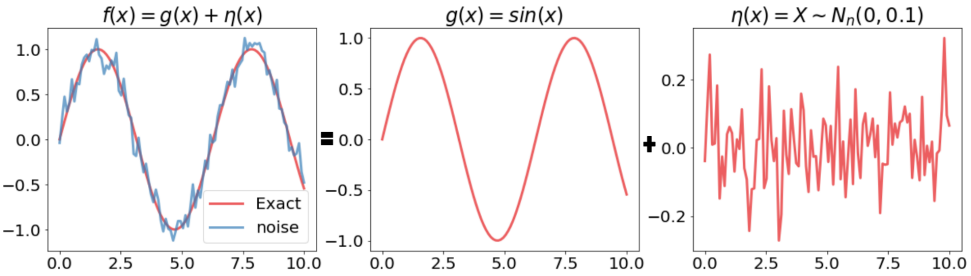


Figure C.1: Superposition of noisy signal

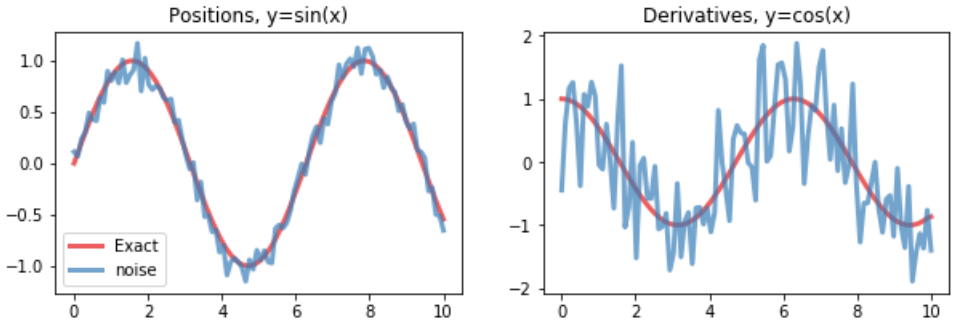


Figure C.2: Comparison of taking the derivative of noisy and clean data with finite difference

Instead, it is proposed use Gaussian processes to first fit the noisy signal, with the aim to reduce the error of the noise and to create a smoother signal. Then the derivative of the Gaussian process is taken to obtain the derivatives of the signal. A review of Gaussian processes and kernels is given in appendix B. To take the derivative of the a Gaussian process, A Gaussian process is first defined as:

$$f(\mathrm{x}) \sim \mathcal{GP}(m(\mathrm{x}), K(\mathrm{x}, \mathrm{x}')) \tag{C.4}$$

Where we can use equations C.5 and C.6 to predict the positions and the uncertainty. Here M denotes test points and N denotes training points.

$$\mu_* = \mathrm{K}_{MN} \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{y} \tag{C.5}$$

$$\Sigma. = \mathrm{K}_M - \mathrm{K}_{A/N} \left(\mathrm{K}_N + \sigma_n^2 \mathrm{I}_N\right)^{-1} \mathrm{K}_{NM} \tag{C.6}$$

If a Gaussian process is differentiable, the derivative of a Gaussian process is a also a Gaussian process [29]. The derivative of a Gaussian process is then defined as:

$$f'(\mathrm{x}) \sim \mathcal{GP}'(\frac{d}{dx}m(\mathrm{x}), \frac{d}{dx}K(\mathrm{x}, \mathrm{x}')) \tag{C.7}$$

The mean and covariance of the differentiated Gaussian process are given by equations C.8 and C.9 respectively [29]. Here $\odot$ represents an element-wise product, μ and Σ are the mean and covariance of the original Gaussian process, $\alpha = K(X,X)^{-1}y$, $\tilde{X}_* = [x_* - x_1, \ldots, x_* - x_N]^T$ and Λ, C_{kk}, C_{xkk} and C_{xkxk} are dependent on the kernel.

$$\mathbb{E}\left[\frac{\partial f_*}{\partial x_*}\right] = \left|\Sigma\Lambda^{-1} + I\right|^{-1/2} (\Sigma + \Lambda)^{-1}\tilde{X}_*^T(\alpha \odot k(X, \mu, \Lambda + \Sigma)) \tag{C.8}$$

$$\begin{aligned}&\mathbb{C}_{x_*}\left[\Lambda_i^{-1}\tilde{X}_*^{(i)T}\left(k\left(X, x_*\right) \odot \alpha\right), \Lambda_j^{-1}\tilde{X}_*^{(j)T}\left(k\left(X, x_*\right) \odot \alpha\right)\right] \\ &\quad = \Lambda_i^{-1}\Lambda_j^{-1}\alpha_k^T\left(X^{(i)}X^{(j)T} \odot C_{kk} - X^{(i)}C_{xkk}^{(j)} - X^{(j)T}C_{xkk}^{(i)} + C_{xkxk}^{(ij)}\right)\alpha_l\end{aligned} \tag{C.9}$$

For a squared exponential kernel, its derivative is given by:

$$\frac{\partial k\left(x_1, x_2\right)}{\partial x_2} = \Lambda^{-1}\left(x_1 - x_2\right)k\left(x_1, x_2\right) \tag{C.10}$$

where Λ is given by equation C.11 and l_i are the characteristic length-scales for each dimension.

$$\Lambda = \begin{bmatrix} l_1^2 & 0 & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & l_D^2 \end{bmatrix} \tag{C.11}$$

The kernel moments needed to calculate the covariance of the derivative are then given by equations C.12 [29]

$$\begin{aligned}C_{kk}\left(\mu, x_1, x_2, \Sigma\right) = &\left|2\Sigma\Lambda^{-1} + I\right|^{-1/2} k\left(x_1/2, x_2/2, \Lambda/2\right)k\left(\left(x_1 + x_2\right)/2, \mu, \Lambda/2 + \Sigma\right) \\ &- \left|\Sigma\Lambda^{-1} + I\right|^{-1} k\left(\mu, x_1, \Lambda + \Sigma\right)k\left(\mu, x_2, \Lambda + \Sigma\right)\end{aligned} \tag{C.12}$$

$$C_{xkk}\left(\mu, x_1, x_2, \Sigma\right) = k\left(\frac{x_1}{2}, \frac{x_2}{2}, \frac{\Lambda}{2}\right)\kappa\left(\mu, \Sigma, \left(x_1 + x_2\right)/2, \Lambda/2\right) - \kappa\left(\mu, \Sigma, x_1, \Lambda\right)\bar{k}\left(\mu, x_2, \Sigma, \Lambda\right) \tag{C.13}$$

$$C_{xkxk}(\mu, x_1, x_2, \Sigma) = k\left(\frac{x_1}{2}, \frac{x_2}{2}, \frac{\Lambda}{2}\right)\bar{k}(\mu, \Sigma, (x_1 + x_2)/2, \Lambda/2)\left[\left(2\Sigma\Lambda^{-1} + I\right)^{-1}\right.$$
$$\left.\left(\Sigma\Lambda^{-1}(x_1 + x_2) + \mu\right)\left(\Sigma\Lambda^{-1}(x_1 + x_2) + \mu\right)^T\left(2\Sigma\Lambda^{-1} + I\right)^{-1} + \Sigma(\Sigma + \Lambda/2)^{-1}\Lambda/2\right.$$
$$-\kappa(\mu, x_1, \Sigma, \Lambda)\,\kappa(\mu, x_2, \Sigma, \Lambda)^T \tag{C.14}$$

C

The derivative of the Gaussian process can be used as a numerically robust differentiator.

D

Implementation of data-driven techniques

D.1. Data-driven validation

If one wants to use symbolic regression tools in an experimental setting, it is still challenging to find out if the found equation is actually the correct one. It becomes even more challenging if multiple equations are found and a single equation has to be chosen. Since the correct equation is not known, one has to revert to data-driven techniques for validation. Standard techniques for measuring the effectiveness of a fit include calculating the coefficient of determination (R^2) on unseen data. The coefficient of determination is a standardised measure that compares what percentage of the predicted data's variation can be explained by the variation of the true data, given the same input. This provides a measure of the quality of a fit. The question left is to determine what part of the data should be reserved for testing purposes. For a comparison between cross-validation and rolling cross-validation, the following scenario is considered, where two Gaussian processes follow identical training procedures. In figure D.1 an example is shown of time series data and a fit subjected to cross-validation. It can be seen that the error is low because the mean and the true signal are very close on the part where there is no data. However, in the case of rolling cross-validation seen in D.2, it can be seen that true signal deviated significantly from the mean of the fit. This effect occurs in time-series data because information from the future leaks to the test data. In the case of comparing equations, these large deviations in score become very useful. Governing equations will generalise better than over-fitting equations that only produce good scores within the training time interval. Therefore rolling cross validation was chosen.

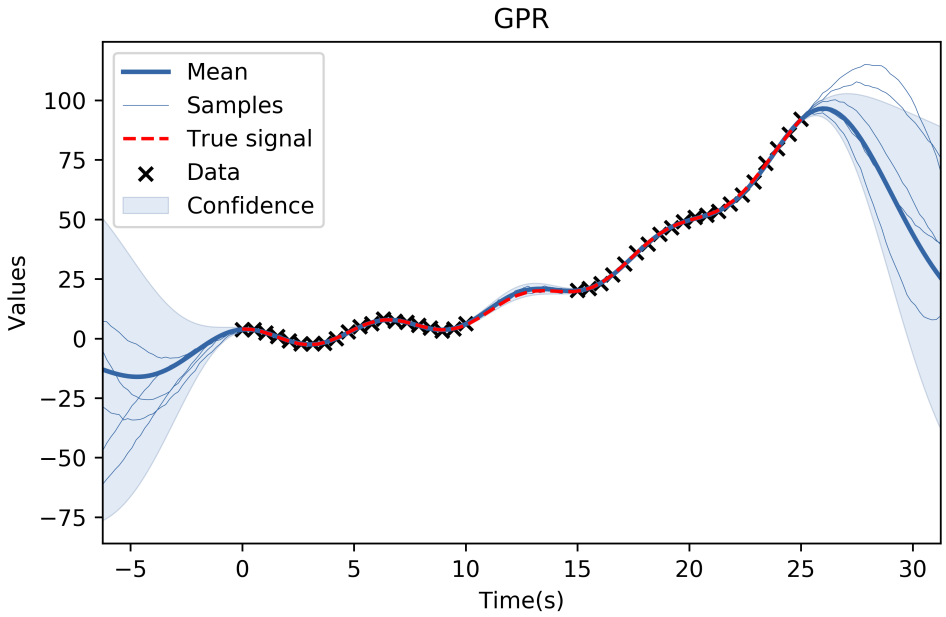


Figure D.1: Gaussian process subjected to cross-validation show future leakage

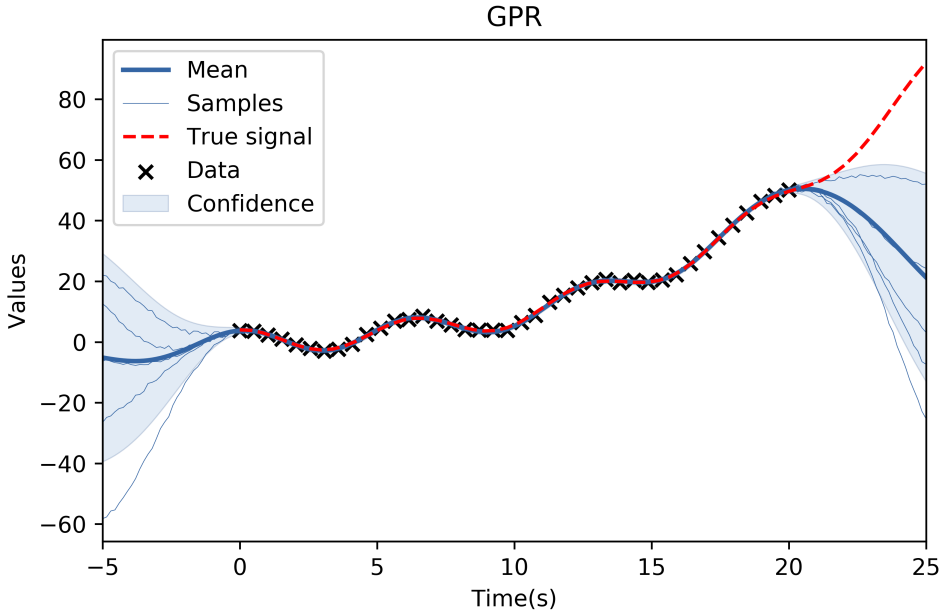


Figure D.2: Gaussian process subjected to rolling cross-validation

The most basic implementation of rolling cross-validation is shown in figure D.3. However, this is still not sufficient because at this point, the solution is non-unique. The SINDy algorithm doesn't output a single equation but multiple equations depending on the regularisation parameters. The SINDy algorithm proposes to write an equation in the form of equation D.1. In this equation Θ is library of candidate functions and Ξ are the parameters of these candidate functions. The noise is modelled as η. In the SINDy algorithm the equation is then rewritten in a sequentially thresholded least squares formulation with L2-regularisation. This is shown in equation D.2. Here α is the first regularisation parameter. This minimisation process is repeated with all the terms that have a parameter value above a threshold τ. This is the second regularisation parameter. On the basis of these two regularisation parameters multiple equations can be found on the same training set. Therefore a more complex scheme is needed to single out an equation and test its score with rolling cross-validation.

$$\dot{X} = \Theta(X)\Xi + \eta \tag{D.1}$$

$$\underset{\Xi \in (-\infty, \infty]}{\arg\min} \left\| \Xi - \Theta(X)^{-1}\dot{X} \right\| + \alpha \|\Xi\|^2 \tag{D.2}$$

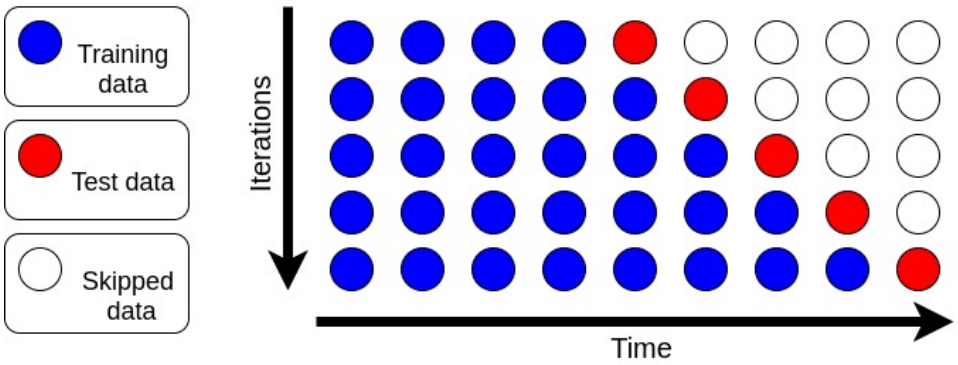


Figure D.3: Rolling cross-validation

In figure D.4 a more extensive method is proposed. Here validation data is added to single out the governing equation, as it tunes the regularization parameters. Data is skipped between the training and validation data because it was found that for symbolic regression is takes a bit longer before the found equations start diverging, as compared to other regression types. With the validation data a Pareto frontier can be created. An example of a Pareto frontier is shown in figure D.5. At the corners of the front line, one cannot improve the predictive ability without sacrificing the simplicity (1/number of nodes). In order to choose between the equations on the Pareto frontier L1-regularization is used. This is done according to equation D.3. Here R^2 is the coefficient of determination of the equation on

validation data, λ is the regularisation parameter and $length$ is the length of the equation. Depending on the value of λ any equation on the Pareto frontier can be found. As a heuristic, it is found that $\lambda = 0.01$ yields good results. This value yiels good results because in general the equation with the highest R^2 is the closest to the true equation. However, sometimes small extra terms are added that fit the noise and thus slightly improves the R^2. A regularisation parameter of $\lambda = 0.01$ is often just enough to prevent these small extra terms from being added. Once an equation is chosen, the average of the test scores can be taken to provide an uncertainty estimate.

$$score = R^2 - \lambda * length \tag{D.3}$$

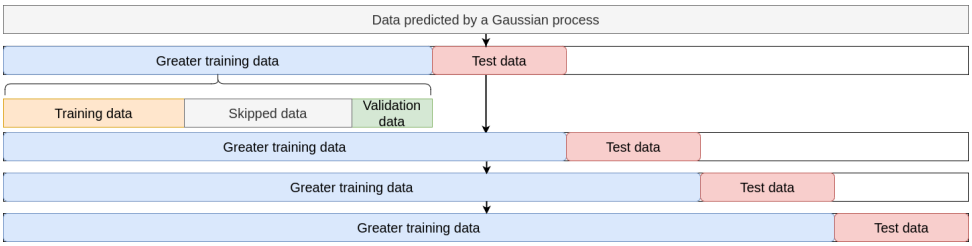


Figure D.4: Proposed training procedure

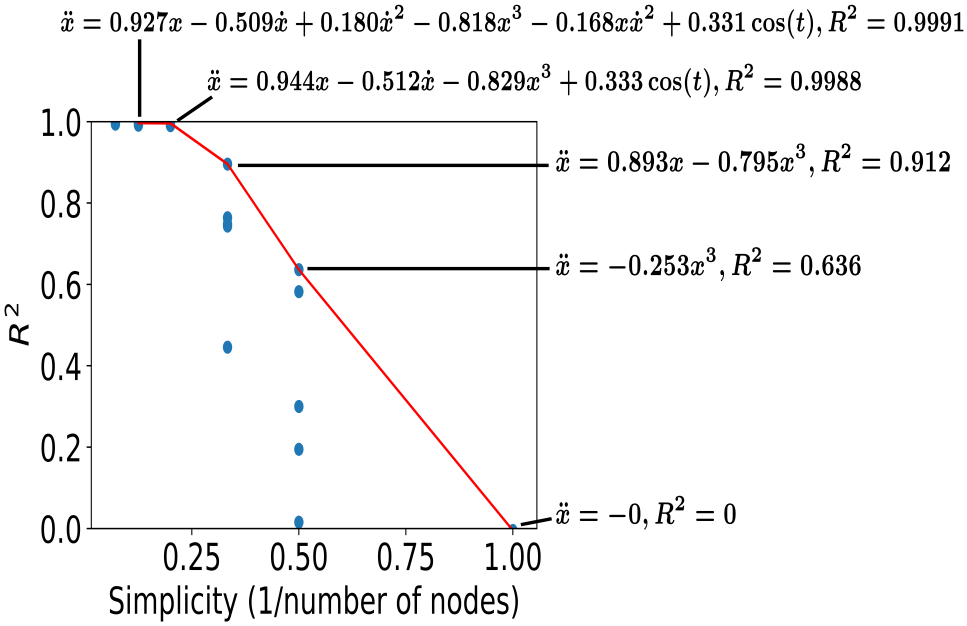


Figure D.5: Example of the Pareto frontier for a Duffing oscillator

E

Additional results

E.1. Found equations with low R^2 scores

In case of failure the algorithm often still leaves us with useful information about the true equation. In addition to the Van der Pol oscillator, a similar effect also occurs for the Lorenz attractor and all the cases of the Duffing oscillator.

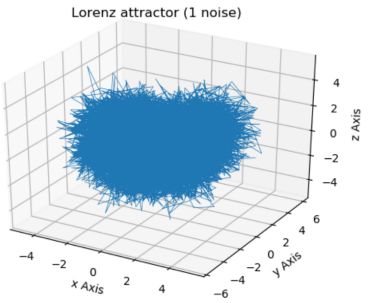
Consider the Lorenz attractor shown in figure E.1. The clean version is shown in figure E.2 and is described by equation E.1. The version in figure E.1 is scaled to have a variance of 1 and has noise with a variance of 1 added to it. After the Gaussian process regression, the mean absolute error is reduced by 78% compared to the clean version. The result is shown in figure E.3.

The Gaussian process model is then used to predict 60.000 point in the same timespan, which is used as an input for SINDy. The combination of SINDy and the described algorithm in section D.1, finds equations E.2 and figure E.4.

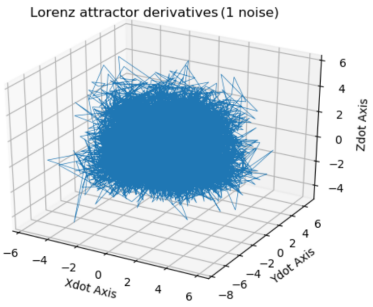
In an experimental setting, it would be hard to know whether the equation is correct, even though the form of the equation is the same. The R^2 is 0.86, which could also occur with an equation with the wrong terms. In this case we know the equation contains the correct terms because we can compare it to the true equation. In an experimental setting this is not possible but it shows that even though the R^2 is quite a bit off from 1, it can still contain the correct terms.

$$\begin{aligned} \dot{x} &= 10y - 10x \\ \dot{y} &= 28x - y - xz \\ \dot{z} &= xy - 2.667z \end{aligned} \tag{E.1}$$

$$\begin{aligned} \dot{x} &= 7.350y - 7.095x \\ \dot{y} &= 20.451x + 1.121y - 0.829xz \\ \dot{x} &= -4.9441 + 0.896xy - 2.115z \end{aligned} \tag{E.2}$$



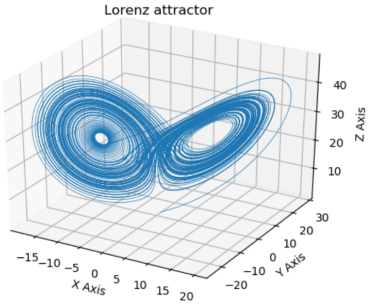
(a) Scaled Lorenz attractor positions with added noise



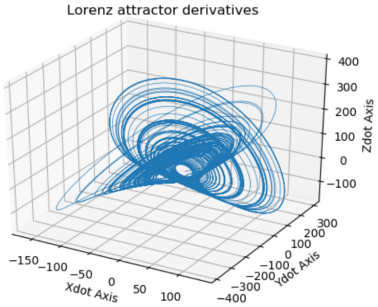
(b) Scaled Lorenz attractor derivatives with added noise

Figure E.1: Scaled Lorenz attractor with added noise

E

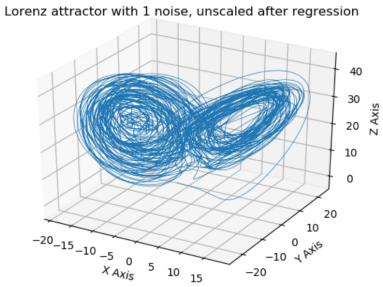


(a) Clean Lorenz attractor

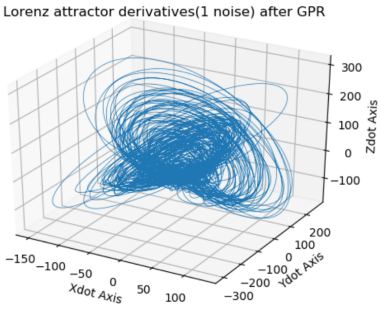


(b) Clean Lorenz attractor derivatives

Figure E.2: Clean Lorenz attractor

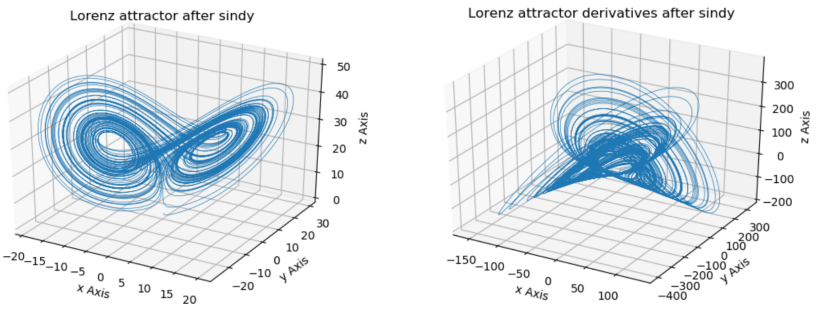


(a) Lorenz attractor after Gaussian process regression



(b) Lorenz attractor derivatives after Gaussian process regression

Figure E.3: Lorenz attractor after Gaussian process regression



(a) Lorenz attractor after SINDy

(b) Lorenz attractor derivatives after SINDy

Figure E.4: Lorenz attractor after SINDy

E.2. Additional systems

The algorithm can be extended to many more systems. In this section a coupled spring-mass system, the Lotka-Volterra model and a double pendulum are taken as additional examples.

E.2.1. Coupled spring-mass system

Here a coupled spring mass system is evaluated where two masses and two springs are connected in series as shown in figure E.5. The data provided for the algorithm has added noise with the same variance as the signal. The positions of the masses are shown in figure E.6, where x_1 and y_1 describe the position of the first mass,and x_2 and y_2 describe the position of the second mass. The clean version of the system can be described by equations E.3 and is shown in figure E.7. The system after Gaussian process regression is shown in figure E.8. The found equations are shown in equations E.4 and the R^2 score is 0.95. Due to the simplicity of the equations and the restricted search, the algorithm can find the correct equations in situations with a high amount of noise.

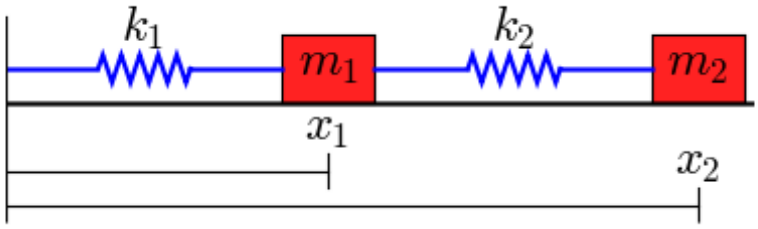


Figure E.5: Coupled spring-mass system, reprinted from scipy documentation,2018, retrieved from https://scipy-cookbook.readthedocs.io/items/CoupledSpringMassSystem.html

$$\begin{aligned} \dot{x_1} &= y_1 \\ \dot{y_1} &= -36 - 48x_1 - 0.8y_1 + 40x_2 \\ \dot{x_2} &= y_2 \\ \dot{y_2} &= 26.67 + 26.67x_1 - 26.67x_2 - 0.33y_2 \end{aligned} \tag{E.3}$$

$$\begin{aligned} \dot{x_1} &= 0.998y_1 \\ \dot{y_1} &= -34.529 - 47.458x_1 - 0.796y_1 + 38.852x_2 \\ \dot{x_2} &= 0.996y_2 \\ \dot{y_2} &= 26.070 + 26.391x_1 - 26.178x_2 - 0.448y_2 \end{aligned} \tag{E.4}$$

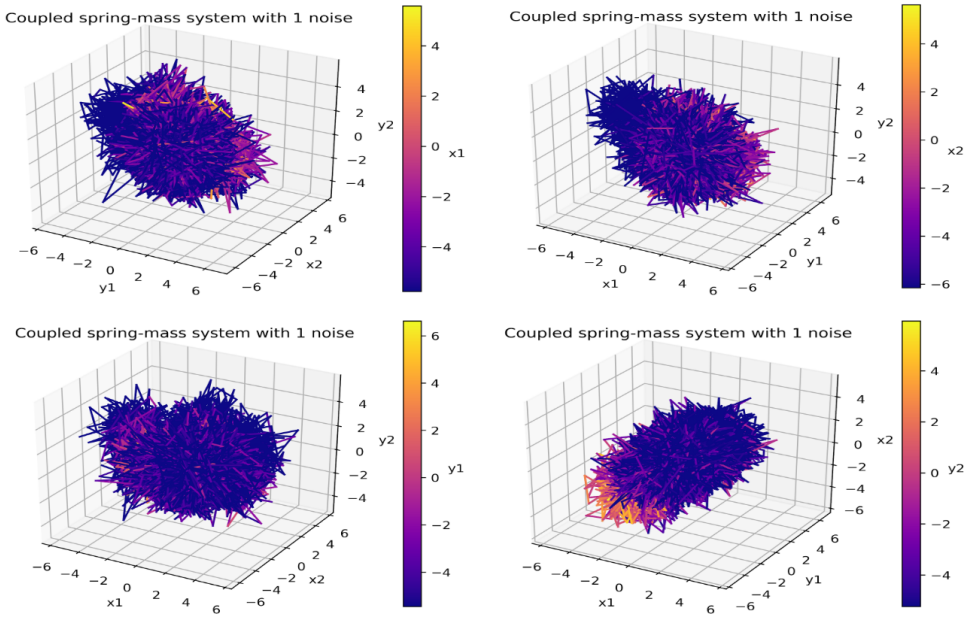


Figure E.6: Noisy coupled spring-mass system

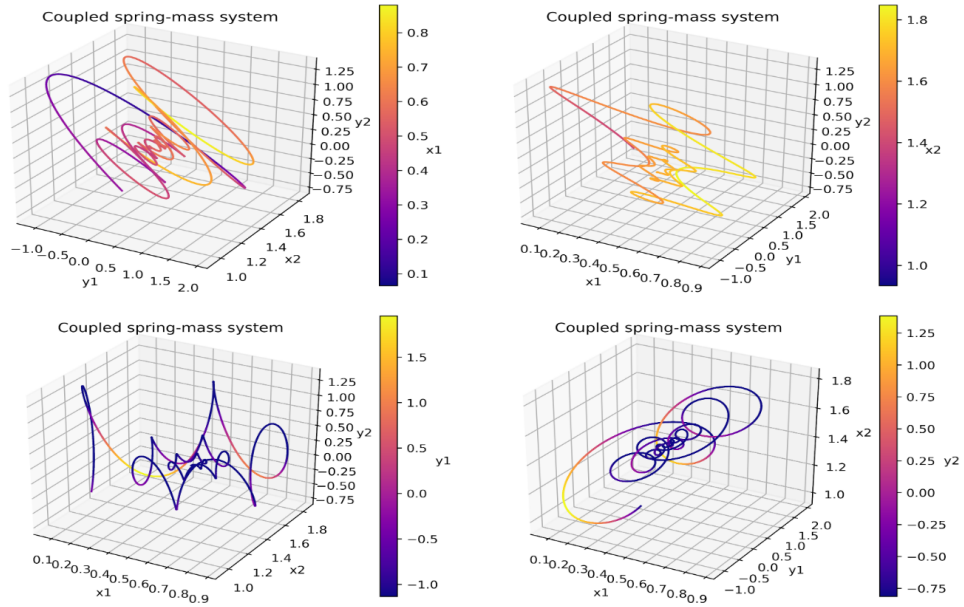


Figure E.7: Clean coupled spring-mass system

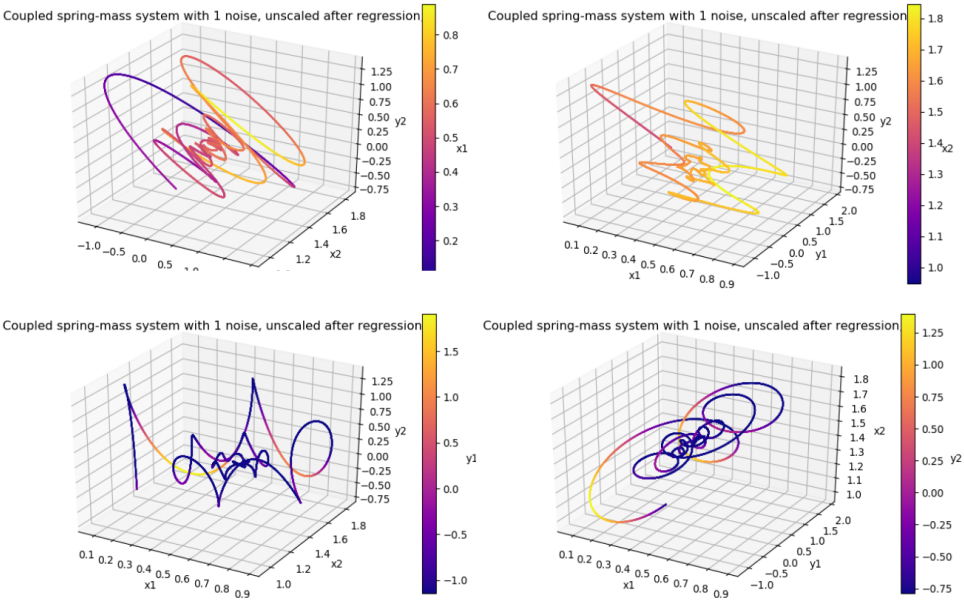


Figure E.8: Coupled spring-mass system after Gaussian process regression

E.2.2. Double pendulum

The equations used to model the outer part of a double pendulum are shown in equations E.5. The algorithm was not able to find these equations because the search was deemed trivial if terms like $\frac{sin(x-2y))}{(cos(x-y))^2}$ are added to the SINDy library as a whole. This would require too much knowledge of what the equation should look like beforehand. It was found that the Gaussian process step still works well. The system is shown in figure E.9, where theta1 represents the angle of the first part of the pendulum and theta2 represents the angle of the second part of the pendulum. The clean version is shown in figure E.10. The system after Gaussian process regression is shown in figure E.11. The Gaussian process is able to reduce the mean absolute error compared to the clean system by 92% and it can be seen that the system after regression is very similar to the clean system.

$$\begin{aligned} \ddot{x} &= 0.5\dot{x}^2 sin(2x-2y) + 2.0\dot{y}^2 sin(x-y) + 24.525 sin(x) + 4.905\frac{sin(x-2y))}{(cos(x-y))^2} - 2 \\ \ddot{y} &= -1.0\dot{x}^2 sin(x-y) - 0.5\dot{y}^2 sin(2x-2y) + 2.4525 sin(y) - 7.3575\frac{sin(2x-y))}{(cos(x-y))^2} - 2 \end{aligned} \tag{E.5}$$

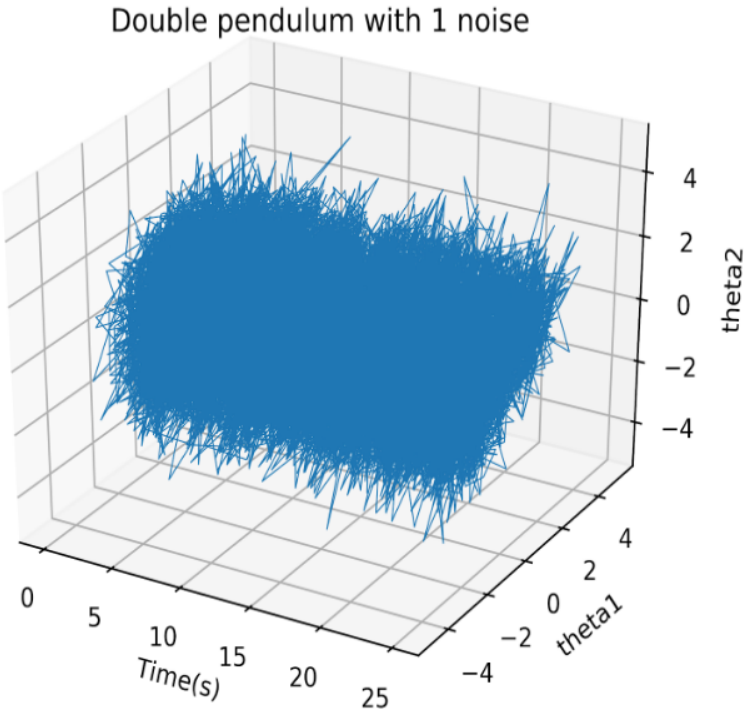


Figure E.9: Noisy movement of the outer part of a double pendulum

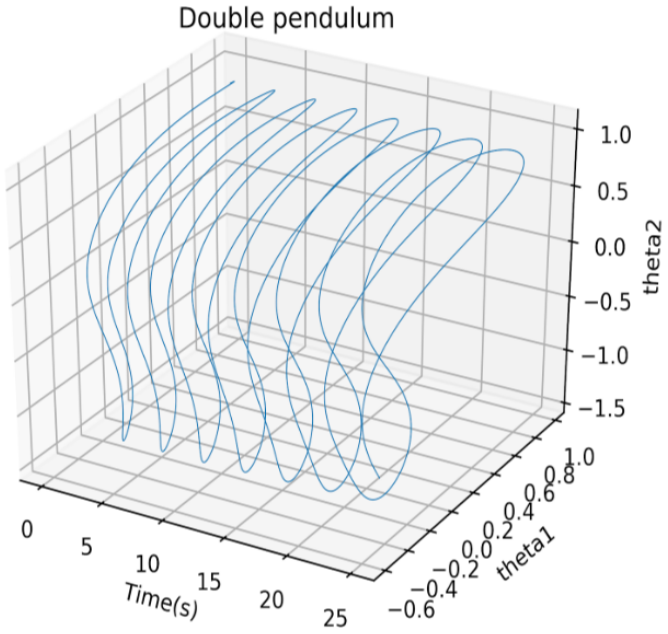


Figure E.10: Clean movement of the outer part of a double pendulum

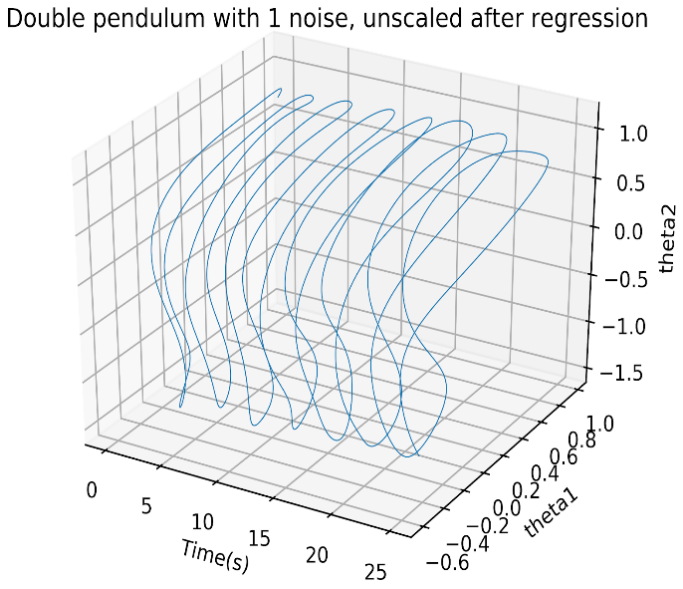


Figure E.11: Movement of the outer part of a double pendulum after Gaussian process regression

F

Codes used

All codes can be found on https://github.com/bessagroup/SymbolicAI.