Data-driven model discovery using Sparse Identification of Nonlinear Dynamics

Kryštof Bystřický*[1]

[1] *Czech Technical University in Prague, Faculty of Mechanical Engineering, Department of Instrumentation and Control Engineering, Technická 4, 166 07 Prague 6, Czech Republic, Krystof.Bystricky@fs.cvut.cz*

Abstract

The problem of obtaining models describing the dynamics of a certain system is current across many industries. The model can be created from the physical principles of the system, this approach however requires a developed theory of the underlying system. An alternative approach is model identification from measured data, which is often the only viable approach for complex systems without a developed theory. In this paper, I apply a method called Sparse Identification of Nonlinear Dynamics (SINDy), which utilizes both approaches, identification from data using at least some limited knowledge of the system. The method will be used to identify a nonlinear model of a pendulum-cart system from data with additive noise. The paper also describes methods for filtering and numerical differentiation of the measured signals and a new method for model selection for sparse models.

Key-words: Machine learning; system identification; sparse regression; dynamical systems; numerical differentiation; spectral methods

1. Introduction

The problem of discovering mathematical models describing real-world phenomena is relevant since the start of the scientific revolution. Having the ability to quantitatively describe reality opens new opportunities in both engineering and sciences. This paper focuses on dynamical models, which are mathematical models that describe the behaviour of a system in time.
The model discovery process is about identifying and describing patterns in the measurements. Traditionally, this relied heavily on expert knowledge and intuition. An expert had to notice the patterns, be able to reduce them into mathematical form and create the mathematical model describing the observations. The emergence of computers, increasing computational power, and advances in machine learning enable an alternative approach. Many machine learning methods, such as neural networks, are capable of describing dynamical systems purely from data. These models are however difficult to interpret (black box models), and they often don't respect physical constraints on the model dynamics. These issues justify the recent interest in physics-informed machine learning, which combines expert knowledge of physics and machine learning.
The identified model then has to be evaluated. According to the philosophy of Occam's razor, also known as the law of parsimony, a good model should be as simple as is necessary to accurately describe the observations. The mathematical model should also be able to generalize to data that haven't been seen during the model creation process. Physics-informed machine learning constrains the space of possible models, which prevents overfitting and therefore promotes the model's generalization capability.

2. Sparse Identification of Nonlinear Dynamics

Sparse Identification of Nonlinear Dynamics [1], or SINDy, is a method that utilizes both expert knowledge and machine learning. The expert's input is in reducing the space of possible models by picking a set of functions that might describe the real system dynamics. Sparsity-promoting regression is then used to create models from measurement data. By promoting sparsity, the functions that turn out to not be relevant in the underlying dynamics will be completely omitted

in the final model. Sparsity therefore means that the final model contains as few functions as possible. This follows the law of parsimony, which in the context of mathematical models means that we should value the simplicity of our models as well as their accuracy.

2.1. Regression for dynamical systems

2.1.1. General formulation

Dynamical systems are described by a system of ordinary differential equations (ODEs). Systems of ODEs are generally described by

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t)). \tag{1}$$

The vector $\mathbf{x}(t) \in \mathbb{R}^d$, where d is the number of state variables, defines the state of the system at time t. The vector function $\mathbf{f}$ describes the effect of natural dynamics at time t. The external input vector $\mathbf{u}(t) \in \mathbb{R}^b$, where b is the number of external inputs, and $\mathbf{g}$ is the vector function describing the effect of external forcing on the system. Note that $\mathbf{g}$ is a function of both state and input, as an input might have different effect on the system depending on its current state.

2.1.2. Matrix-vector formulation

For the SINDy [1] algorithm to be directly applicable, every single ODE $\dot{x}_i$ from $\dot{\mathbf{x}} = [x_1, \ldots, x_d]^\mathbf{T}$ must be expressible as a linear combination of functions of $\mathbf{x}$ and $\mathbf{u}$

$$\dot{x}_i(t) = \xi_1\theta_1(\mathbf{x}(t), \mathbf{u}(t)) + \ldots + \xi_m\theta_m(\mathbf{x}(t), \mathbf{u}(t)) \tag{2}$$

where ξ are scalar parameters and m is the total number of candidate functions. Simplifying the notation so that $\theta_i(t) = \theta_i(\mathbf{x}(t), \mathbf{u}(t))$, the equation (1) can be formulated in vector notation as

$$\dot{x}(t) = \begin{bmatrix}\theta_1(t) & \ldots & \theta_m(t)\end{bmatrix} \begin{bmatrix}\xi_1 \\ \vdots \\ \xi_m\end{bmatrix} = \boldsymbol{\Theta}\boldsymbol{\xi} \tag{3}$$

where the matrix $\boldsymbol{\Theta}$ is the set of candidate functions and the vector $\boldsymbol{\xi}$ is the vector of coefficients of those functions. The regression task is therefore to find a vector of coefficients ξ that represents $\dot{x}_i$ using functions from $\boldsymbol{\Theta}$.
Because the continuous functions $\theta(t)$ are only an abstraction that cannot be worked with numerically, we must approximate them with a finite number of measurements $\theta[k]$, where k is the time-sample index. Note that $\theta[k]$ are vectors of *measurements*, so they're far from a perfect substitute for the actual functions $\theta(t)$. Whether measurements $\theta[k]$ represent the underlying function $\theta(t)$ well doesn't depend only on the sampling frequency or the number of samples, but also on the way they've been generated. If, for example, we had a function $\theta(t) = x_1 \sin(x_2) + 1$, but the state variable x_2 was kept constant at 0 during the experiment, then the set of measurements $\theta[k]$ would be a terrible representative of $\theta(t)$, because it wouldn't describe the effects of any of its variables x_1 and x_2.
The approximation of (3) therefore has the form

$$\dot{x}_i[k] = \begin{bmatrix}| & \ldots & | \\ \theta_1[k] & \ldots & \theta_m[k] \\ | & \ldots & |\end{bmatrix} \begin{bmatrix}\xi_1 \\ \vdots \\ \xi_m\end{bmatrix} = \boldsymbol{\Theta}(\mathbf{X}, \mathbf{U})\ \boldsymbol{\xi}. \tag{4}$$

$\boldsymbol{\Theta}(\mathbf{X}, \mathbf{U}) \in \mathbb{R}^{N\times m}$ is called the *function library*, it has m columns representing the candidate functions, and N rows representing time-samples. The columns $\theta_i[k]$ representing the candidate functions are computed from state measurements $\mathbf{X}$ and input measurements $\mathbf{U}$. The state derivative measurements $\dot{\mathbf{X}}$ must be computed from the state measurements $\mathbf{X}$ numerically, using for example spectral differentiation described later in this paper.
The original paper [1] didn't mention identification of systems with external inputs $\mathbf{u}$ as in the case above. Another paper was released shortly after, where an extension named SINDYc [2] included the control input $\mathbf{u}$ in the function library. The extension essentially doesn't make a distinction between the inputs $\mathbf{u}$ and states $\mathbf{x}$, they can be treated equally with no extra cost. The inputs $\mathbf{u}$ will not appear in the equations later in the paper, since adding them is trivial.

2.2. Sparse regression

2.2.1. Description

The equation (4) represents the general regression problem

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{5}$$

The objective is to find a solution vector $\mathbf{x}$ (not to be confused with the state vector $\mathbf{x}$), or $\boldsymbol{\xi}$ in the ODE formulation. There are many algorithms for solving this problem, many of them put some constraints on the solution $\boldsymbol{x}$. Following the law of parsimony, we want to find a solution $\mathbf{x}$ that is *sparse*, meaning it has as few non-zero elements as possible. Note that when a solution coefficient ξ_i is 0, its respective candidate function $\theta_i(t)$ isn't present in the final model.

2.2.2. Sequentially thresholded least squares

The sequentially thresholded least squares (STLS) algorithm used in the original SINDy [1] paper uses standard least-squares regression

$$\arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 \tag{6}$$

and then sets all elements $x \in \mathbf{x}$ that are below a defined hyperparameter threshold λ to 0. These two steps are then repeated multiple times until the solution $\mathbf{x}$ no longer changes between iterations. In the ODE formulation (4), the algorithm sequentially reduces the number of considered candidate functions θ. On the first iteration, it works with the full function library $\boldsymbol{\Theta} \in \mathbb{R}^{N\times m}$ and produces a non-sparse solution $\boldsymbol{\xi}$, whose elements ξ_i which are below the threshold λ are set to 0. If an element ξ_i is set to 0, its respective candidate function θ_i is dropped from $\boldsymbol{\Theta}$ and won't be considered in the next iterations. The number of candidate functions usually drops significantly after the first iteration. The condition number κ, representing the well-posedness of the regression problem, is defined as a ratio of the biggest and the smallest singular value

$$\kappa = \frac{\max \sigma(\boldsymbol{\Theta})}{\min \sigma(\boldsymbol{\Theta})} \tag{7}$$

The condition number κ, is very high at the first iteration, meaning the problem is ill-posed and the solution isn't reliable. In the later iterations, as columns of $\boldsymbol{\Theta}$ are dropped, κ also usually decreases to acceptable values. The STLS method produces sparse solutions $\mathbf{x}$ while preserving the numerical robustness and low time complexity of least-squares algorithms.

2.2.3. Sequentially energy thresholded least squares

The STLS algorithm sequentially drops candidate functions θ_i based on their respective parameter values ξ_i under the assumption that low values of ξ_i indicate that the respective candidate function θ_i has a small effect on the target variable. However, when working with unnormalized measurements $\theta[k]$, this isn't necessarily true. Interpreting the measurements $\theta[k]$ as signals, a high energy signal $\theta_i[k]$ with a low coefficient ξ_i might have a higher effect than another signal $\theta_j[k]$ with a relatively higher coefficient ξ_j. To deal with this issue, I modified the STLS algorithm so that the thresholding is done based on the signal's total implied energy, defined for every candidate function and parameter vector $(\theta[k], \xi)$ pair as

$$E(\xi, \theta[k]) = \sum_{k=0}^{N-1} (\xi\, \theta_k)^2. \tag{8}$$

This energy E is calculated for every candidate function present in the function library, the highest energy $\max\{E(\xi, \theta[k])\}$ then becomes a baseline from which the threshold λ is calculated as

$$\lambda = \max\{E(\xi, \theta[k])\} \cdot \lambda_R \tag{9}$$

where λ_R is a hyperparameter setting the relative energy ratio between the lowest acceptable energy signal and the maximum energy signal. The candidate functions θ whose implied energies are lower than λ are then dropped from $\boldsymbol{\Theta}$ and their respective coefficients ξ set to 0.

2.3. Implicit SINDy

According to the equation (1), the standard SINDy method cannot be used if the dynamical system is described by a rational function in the general form

$$\dot{\mathbf{x}}(t) = \frac{\mathbf{f}(\mathbf{x}(t))}{\mathbf{h}(\mathbf{x}(t))}. \tag{10}$$

In this case, the dynamics cannot be reduced into a linear combination of functions as before in (2). An extension called implicit-SINDy [3] can deal with this problem. The system of ODEs is transformed into an implicit form by multiplying both sides by the function $\mathbf{h}$

$$\dot{\mathbf{x}}(t)\ \mathbf{h}(\mathbf{x}(t)) = \mathbf{f}(\mathbf{x}(t)) \tag{11a}$$

$$0 = \mathbf{f}(\mathbf{x}(t)) - \dot{\mathbf{x}}(t)\ \mathbf{h}(\mathbf{x}(t)) \tag{11b}$$

For a single ODE $\dot{x}_i$ from $\dot{\mathbf{x}}$, the equation (10) can again be simplified by using a single symbol θ for all right-hand side functions f, $\dot{x}$ and h.

$$0(t) = \xi_1\theta_1(\mathbf{x}(t)) + \ldots + \xi_m\theta_m(\mathbf{x}(t)) \tag{12}$$

In this formulation, the equation can be again transformed into matrix-vector notation as

$$0(t) = \begin{bmatrix}\theta_1(t) & \ldots & \theta_m(t)\end{bmatrix} \begin{bmatrix}\xi_1 \\ \vdots \\ \xi_m\end{bmatrix} = \boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})\boldsymbol{\xi} \tag{13}$$

One issue now is that our target function $\dot{x}$, which we want to model, now has to be embedded within the function library $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})$. This means that even if we had the solution $\boldsymbol{\xi}$, it wouldn't be as trivial to reconstruct $\dot{x}$ as before, where we simply had to multiply each candidate function by its coefficient and sum the functions. The reconstruction step essentially requires doing the step equations (10), (11a) and (11b) in reverse and the specific operations depend on the chosen structure of the candidate functions. In practice, this can be automated using symbolic math software, in the case of this paper using the SymPy [4] package for Python. Another problem is that the standard regression problem $\mathbf{Ax} = \mathbf{b}$ changed into the homogeneous problem

$$\mathbf{Ax} = \mathbf{0}. \tag{14}$$

The algorithms for solving (14) aren't as robust as for the more general problem (5). The existing methods are very sensitive to noise, making this extension by itself unpractical for real-world applications, where measurement noise is unavoidable.

2.4. Parallel implicit SINDy

Another extension, called SINDy-PI [5], deals with the high sensitivity of implicit-SINDy by picking one of the columns θ_i from $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})$ and moving it to the left-hand side, effectively transforming the problem back into the more general $\mathbf{Ax} = \mathbf{b}$ formulation. If the chosen candidate function θ_i is in the real target dynamics, the solution $\boldsymbol{\xi}$ will be sparse and the model it generates will likely be accurate. The problem (13) is transformed, using a guess θ_i, into

$$\theta_i[k] = \begin{bmatrix}\theta_1[k] \\ \vdots \\ \theta_{i-1}[k] \\ \theta_{i+1}[k] \\ \vdots \\ \theta_m[k]\end{bmatrix}^{\mathrm{T}} \begin{bmatrix}\xi_1 \\ \vdots \\ \xi_{i-1} \\ \xi_{i+1} \\ \vdots \\ \xi_m\end{bmatrix} = \boldsymbol{\Theta}^i(\mathbf{X}, \dot{\mathbf{X}})\,\boldsymbol{\xi}^i \tag{15}$$

where $\boldsymbol{\Theta}^i(\mathbf{X}, \dot{\mathbf{X}})$ is the function library without the i-th function (column) and $\boldsymbol{\xi}^i$ is the solution vector without the i-th coefficient.

The problem (15) is solved for many different left-hand side guess functions θ_i, with different hyperparameter values λ_R for each run. This generates a very large number of models, and requires techniques to sort through them to pick the best ones. When selecting the models, I utilize the fact that when the real model contains some different candidate functions $\theta_i, \theta_j, \theta_k$, then it will likely be identified whenever one of those functions acts as the left-hand side guess. A model that appears consistently therefore is likely to be correct.
Because the method relies on guessing and has to generate a lot of models, it's significantly more computationally difficult. However, the identification method is highly parallelizable, since each model can be identified separately. This high parallelization capability is why the method is called *Parallel* Implicit.

3. Data preprocessing

The SINDy method requires measurement data to identify the dynamics. The dynamics of every generalized coordinate in a mechanical dynamical system are, as a consequence of Newton's second law, descibed by second-order ODEs. When described in the state-space representation, this leads to two state-space variables for generalized coordinate, one for its current value, one for its rate of change. In this paper, I'll assume that we're measuring only the value of the generalized coordinate. The goal of modeling is to find a model for the acceleration. In other words, the acceleration is the target variable. To train the models, we need to estimate these target variables from data. Both velocities and accelerations can be obtained from position measurements using numerical differentiation.

3.1. Spectral differentiation

Spectral differentiation uses the properties of Fourier transforms to compute derivative estimates from measured data [6]. Because we're working with discrete signals and Fourier transforms are a just a mathematical abstraction, we must use Discrete Fourier Transforms (DFTs) instead. DFTs are usually computed using the Fast Fourier Transform (FFT) algorithm, because of its low time complexity.
Assuming a measured discrete signal $x[k]$, its representation in the frequency domain is

$$\hat{x}[v] = \mathrm{DFT}\{x[k]\}, \tag{16}$$

where v is known as the *wavenumber*. The frequency (in $\frac{\mathrm{rad}}{\mathrm{s}}$) ω can be computed from v as

$$\omega[v] = \frac{2\pi\, v}{\Delta t} \tag{17}$$

where Δt is the sampling period. The convenient property of Fourier transforms is that

$$\mathcal{F}\{\dot{x}(t)\} = i\omega\hat{x}(\omega), \tag{18}$$

meaning the Fourier transform of the function derivative in the time domain is equal to the function itself in the frequency domain multiplied by $i\omega$. Using this property, we can estimate the time-derivative of $x[k]$ as

$$\mathrm{DFT}\{\dot{x}[k]\} = i\omega\hat{x}[\omega], \tag{19}$$

which can be transformed back into the time domain using inverse-DFT

$$\dot{x}[k] = \mathrm{iDFT}\{i\omega\hat{x}[\omega]\}, \tag{20}$$

where $\dot{x}[k]$ is the estimate of the function derivative.
Notice that the property 18 implies that the time-derivative of x is the function itself in the frequency domain passed through a high-pass filter defined by $i\,\omega$. When our signals are measurements of some physical process, the signal itself has most of its energy in the lower frequencies, while (white) noise has the same energy at all frequencies. This unfortunately means that numerical differentiation decreases the signal-to-noise ratio, which raises requirements on low sensor noise and filtering methods.

3.2. Spectral filtering

An ideal filter has a gain of 1 in the specified frequency range and a gain of 0 outside the range. Traditional filters cannot meet these demands. Spectral filtering techniques use the FFT to calculate the signal's representation in the frequency domain. When we have this representation, we can simply set all the high-frequency coefficients to 0 and then use inverse-FFT to reconstruct the signal in the time domain. Mathematically, setting frequencies outside some defined range is equivalent to multiplying the frequency-domain signal with a square function $f_{sq}(\omega)$.

$$\hat{x}[\omega] = \mathrm{DFT}\{x[k]\} \tag{21a}$$

$$\hat{x}_f[\omega] = f_{sq}(\omega)\,\hat{x}[\omega] \qquad f_{sq}(\omega) = \begin{cases} 1, & \text{if } \omega \leq \omega_{cutoff}. \\ 0, & \text{if } \omega > 0. \end{cases} \tag{21b}$$

$$x_f[x] = \mathrm{iDFT}\{x_f[\omega]\} \tag{21c}$$

When designing filters, the most important design choice is the cutoff frequency. If it's set too low, then we filter out useful information from the signal, but if it's set too high, we don't get rid of the noise. In this thesis, I choose the cutoff frequency based on the signal's periodogram, which is an estimate of the power spectrum density (PSD).
Any signal can be interpreted as a sum of information and noise. Information is the part of the signal that is generated by the process we intend to actually measure. Noise is the part of the signal that's generated by other processes. Many processses of interest generate information signals that are mostly dispersed in the lower frequencies, or in other words, have a relatively small bandwidth. The noise, on the other hand, typically has a far bigger bandwidth. Noise is often (mathematically) modeled as a white noise. White noise, by definition, has infinite bandwidth and a constant power spectrum density. In practice, the noise isn't as "flat" in the power spectrum as an ideal white noise. Instead, it's distributed with some variance around some mean power value. I choose the cutoff frequency automatically from the periodogram, assuming the noise component of the signal is distributed evenly in the power spectrum. I calculate the noise ceiling,the power that should be higher than than noise's power at most frequencies, by calculating the mean power μ and standard deviation σ of the 50% highest frequency components in the signal. The noise ceiling is then defined as $\mu + z\sigma$, where z is a manually set parameter.
Then I smoothed the periodogram using a simple moving average filter and find the lowest frequency at which the measurement signal crosses below the previously calculated noise ceiling. If the signal is intended to be used for numerical differentiation, it should be "over-filtered", meaning the cutoff frequency should be set much lower than the frequency at which the noise ceiling is reached. By defining the cutoff frequency as the frequency at which the noise becomes more powerful than the signal, the high frequency components of the filtered signal would have a signal-to-noise ratio of roughly 1. For numerical differentiation, this isn't good, because it acts as a high-pass filter, and these low SNR high frequencies would therefore throw off the derivative estimates.
The filter settings from the periodograms are shown in the Figure 1. In the first one x_1, the cutoff frequency is set as the frequency at which the periodogram crosses the threshold (noise ceiling). In the second one, the cutoff frequency is manually offset to the lower frequencies to over-filter the signal for numerical differentiation.

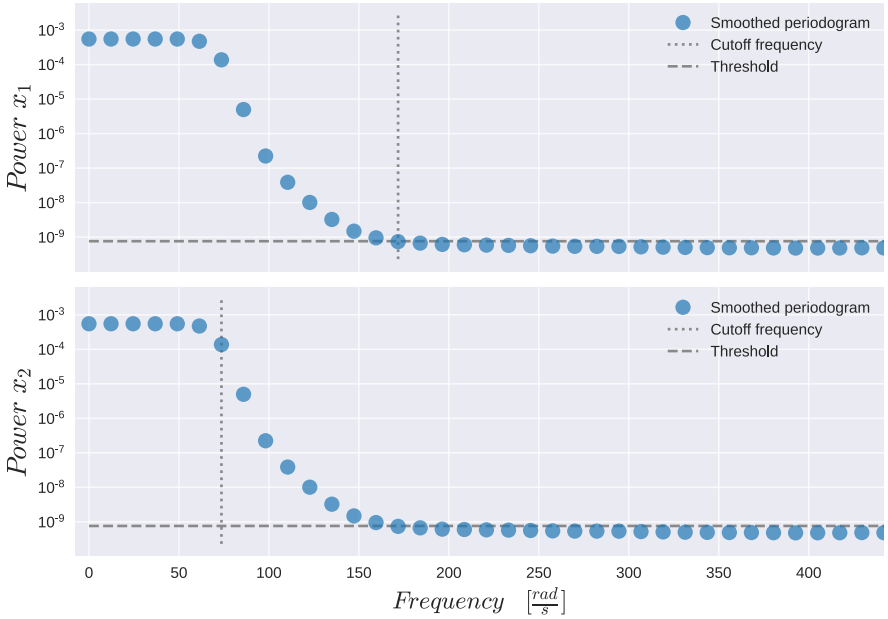


Fig. 1. *Cutoff frequency set at the noise ceiling vs. cutoff frequency set before the noise ceiling - "overfiltering". The threshold represents the noise ceiling, a power value that should be above all noise component power values.*

4. Identification of a pendulum-cart model

4.1. Analytical model and simulation

I'm using the SINDy method to identify the nonlinear dynamical model describing the dynamics of a pendulum mounted on a moving cart. The input u into the system is a force acting on the cart. The position of the cart is given by the first state variable x_1, and the pendulum angle is given by x_2. The other two state variables, x_3 and x_4, are the cart velocity and pendulum angular velocity respectively.
I derived an analytical model of the system using Euler-Lagrangian mechanics according to the paper [7]. The model also contains non-conservative friction forces for both the cart and the pendulum. The friction force is a function of velocity in both cases. The cart acceleration and the angular acceleration of the pendulum are given by $\ddot{x}_3$ and $\ddot{x}_4$ respectively. These are our target variables, which we want to predict using the state variables and inputs and functions derived from them. The analytical (reference) model is used for simulations that generate data, the objective is to reconstruct the reference model from that data. The two second order ODEs describing the dynamical system are

$$\dot{x}_3 = \frac{A(\mathbf{x}, u)}{B(\mathbf{x}, u)} \tag{22a}$$

$$\begin{aligned} A(\mathbf{x}, u) = {} & I_1\, u(t) + {a_1}^2\, m_1\, u(t) - I_1\, b_c\, x_3(t) + +{a_1}^3\, {m_1}^2\, \sin(x_2(t))\, x_4^2(t) + \\ & -{a_1}^2\, b_c\, m_1\, x_3(t) + +a_1\, b_1\, m_1\, \cos(x_2(t))\, x_4(t) + +{a_1}^2\, g\, {m_1}^2\, \cos(x_2(t))\, \sin(x_2(t)) + \\ & + I_1\, a_1\, m_1\, \sin(x_2(t))\, x_4^2(t) \end{aligned} \tag{22b}$$

$$B(\mathbf{x}, u) = -{a_1}^2\, {m_1}^2 \cos^2(x_2(t)) + +{a_1}^2\, {m_1}^2 + m_c\, {a_1}^2\, m_1 + I_1\, m_1 + I_1\, m_c \tag{22c}$$

$$\dot{x}_4 = \frac{C(\mathbf{x}, u)}{D(\mathbf{x}, u)} \tag{23a}$$

$$\begin{aligned} C(\mathbf{x}, u) = {} & b_1\, m_1\, x_4(t) + b_1\, m_c\, x_4(t) + {+}a_1\, g\, {m_1}^2\, \sin(x_2(t)) + a_1\, m_1\, \cos(x_2(t))\, u(t) + \\ & + a_1\, g\, m_1\, m_c\, \sin(x_2(t)) + {+}{a_1}^2\, {m_1}^2\, \cos(x_2(t))\, \sin(x_2(t))\, x_4^2(t) + \\ & - a_1\, b_c\, m_1\, \cos(x_2(t))\, x_3(t) \end{aligned} \tag{23b}$$

$$D(\mathbf{x}, u) = -{a_1}^2\, {m_1}^2 \cos^2(x_2(t)) + {+}{a_1}^2\, {m_1}^2 + m_c\, {a_1}^2\, m_1 + I_1\, m_1 + I_1\, m_c \tag{23c}$$

where I_1 are the pendulum's moment of inertia around its center of mass, a_1 is the distance from the pendulum's joint to its center of mass, b_c and b_1 are the viscous friction coefficients for the cart and pendulum respectively and m_c and m_1 are the masses of the cart and the pendulum. In the simulations, the physical parameters are defined according to the Table 1.

Table 1. *The physical parameters of the simulated pendulum-cart system in base units.*

Physical parameter	Meaning	Value
I_1	Pendulum's moment of inertia around its center of mass	$0.0227\,\mathrm{kg\,m^2}$
a_1	Distance from the pendulum joint to its center of mass	$0.18\,\mathrm{m}$
b_c	Linear cart friction coefficient	$10\,\frac{\mathrm{N\,s}}{\mathrm{m}}$
b_1	Pendulum joint friction coefficient	$0.15\,\frac{\mathrm{N\,s}}{\mathrm{m}}$
m_c	Mass of the cart	$0.8\,\mathrm{kg}$
m_1	Mass of the pendulum	$1\,\mathrm{kg}$
g	Gravitational field intensity	$9.81\,\frac{\mathrm{N}}{\mathrm{kg}}$

The state derivatives $\dot{x}_1$ and $\dot{x}_2$ are equal to the state variables x_3 and x_4 respectively. With the state derivative vector $\dot{\mathbf{x}}$ defined, the system can be numerically simulated using an ODE solver, in this case I used MATLAB's ODE45. During the simulation, the input U was defined as a band-limited random noise process, which was created by low-pass filtering a white noise signal. The result of the simulation are trajectories, or state measurements, $\mathbf{X}$, with sampling period $\Delta t = 0.001\,\mathrm{s}$ and total duration $65\,\mathrm{s}$. To simulate real measurements, I only take the first two variables, the cart position x_1 and the pendulum angle x_2. To simulate noise, I add an additive white noise with zero mean and standard deviation $\sigma = 0.005\mathrm{m}$ for the cart position measurements and $\sigma = 0.0025\mathrm{rad}$ to the angle measurements.

4.2. Processing the measurements

The SINDy-PI method requires full state measurements $\mathbf{X}$, its derivatives $\dot{\mathbf{X}}$, and a function library $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})$. The state variables x_3 and x_4 are equal to $\dot{x}_1$ and $\dot{x}_2$ respectively and represent the cart's linear velocity and the angle's angular velocity respectively. The state derivative variables $\dot{x}_3$ and $\dot{x}_4$ are the cart's linear acceleration and the pendulum's angular acceleration. Before creating the candidate function library, the velocities and accelerations must be estimated from the position measurements x_1 and x_2. This can be done by spectral differentiation. Because the measurements contain noise, they must first be filtered. Since the data is intended for numerical differentiation, the cutoff frequency is set relatively low to over-filter the signal for reasons established before. The filtered signals are then numerically differentiated, generating the velocities $\dot{x}_1 = x_3$ and $\dot{x}_2 = x_4$, which are again differentiated generating $\dot{x}_3$ and $\dot{x}_4$. The signals x_1 and x_2 are shown in Figure 2, and the accelerations $\dot{x}_3$ and $\dot{x}_4$ are shown in Figure 3.

Fig. 2. *Comparison between the original noisy measurements of x_1 and x_2 and the filtered signals.*

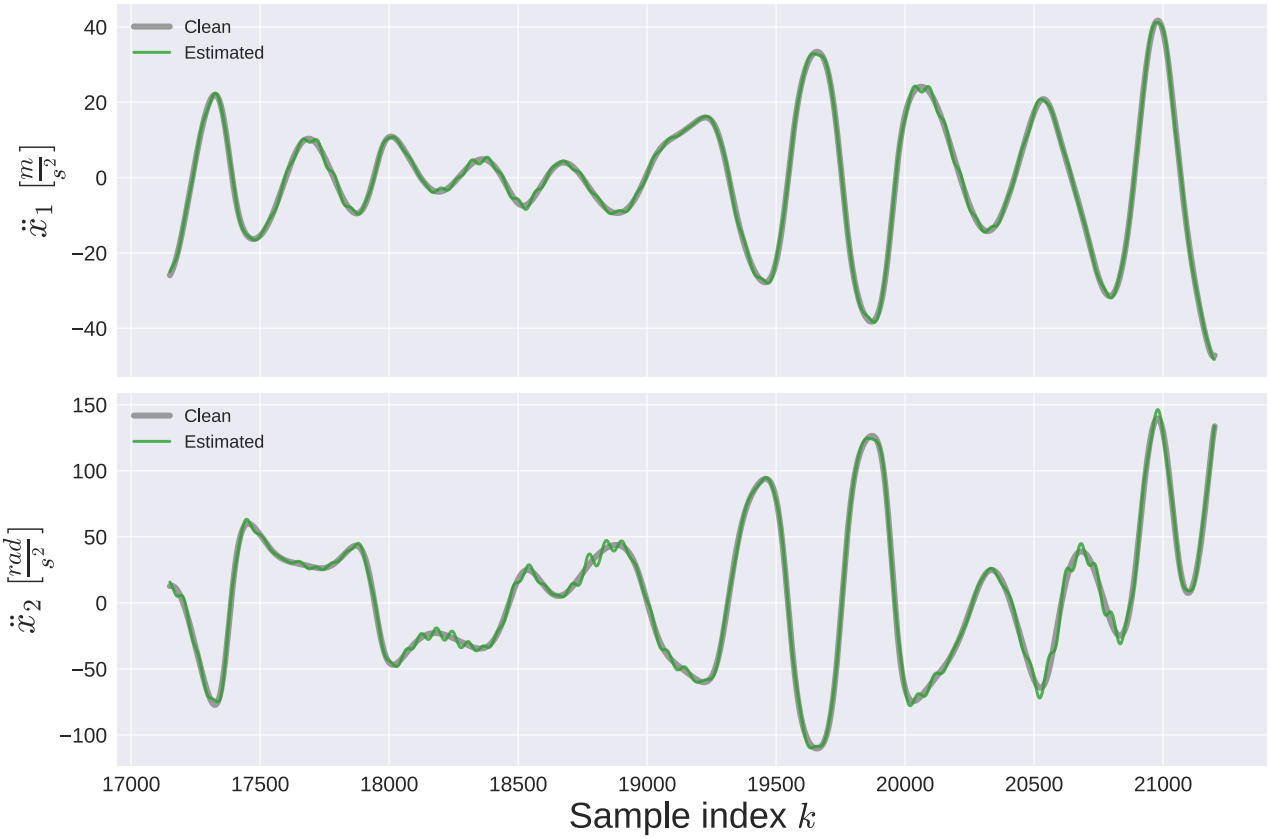


Fig. 3. *Comparison between the computed accelerations $\dot{x}_3$ and $\dot{x}_4$ and the real accelerations.*

4.2.1. Creating the function library

The function library $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})$ is a matrix with columns representing the candidate functions $\boldsymbol{\theta}$. This step is where an expert's input is necessary, because the candidate functions must be created from the positions, velocities and accelerations so that all the terms that are in the real system dynamics are also in the function library. I create these functions by first creating a set of basis functions from which all the other candidate functions are generated. The state variable x_2, the angle of the pendulum, implies rotation, so I added the functions $\sin x_2$ and $\cos x_2$. The positions of the cart x_1 and the angle of the pendulum x_2 don't appear by themselves in the real dynamics, so they're not included in the basis function set. The state derivative variables $\dot{x}_1$ and $\dot{x}_2$ are also omitted, because they're already represented by x_3 and x_4 respectively.
The basis function set is therefore

$$\begin{aligned}\mathbf{Y} &= \{x_3, x_4, \sin(x_2), \cos(x_2), u, 1, \dot{x}_3, \dot{x}_4\} = \\ &= \{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8\}\end{aligned} \tag{24}$$

A large number of candidate functions is then generated from $\mathbf{Y}$ by creating all the possible 4-th order terms. Because I included the constant term $1(t)$ in the basis function set, all 4-th order terms of $\mathbf{Y}$ will actually be all terms of the 1-st order up to the 4-th order. This generates 330 candidate functions, but most of them are be dropped according to a set of manually defined rules, because they're not physically interpretable.
Creating the candidate functions manually is also possible, but it's much more time consuming than creating a large number of candidate functions and then defining a set of rules which are used to remove the bad ones. Including a candidate function that *isn't* in the real system dynamics is no big deal, because sparse regression will simply discard it when looking for a solution. But when a candidate function that is in the real system dynamics is *missing* from the function library, then the solution cannot be found. For this reason, it's safer to create more candidate functions than is necessary than to risk not including an important candidate function.
After dropping bad candidate functions, the function library $\boldsymbol{\Theta}$ has 35 functions left. The reference model equations both have less than 10 unique terms in total, so the solution should pick less than 10 terms from the function library.
It's also important to look at the correlations between candidate functions, because high pair-wise correlations make the regression problem ill-posed. The correlation matrix for the function matrix $\boldsymbol{\Theta}$ used for identifying the model for $\dot{x}_3$ is shown in Figure 4.

High pair-wise correlations themselves might give valuable insight about the actual system. If a time-derivative of some variable is perfectly correlated with another variable, that variable can be used as a model itself. In any case, the correlation matrix serves the expert as a guide when dropping candidate functions from the function library.

4.3. Creating candidate models

The SINDy-PI method described in subsection 2.4. relies on guesses that a function θ_i from $\boldsymbol{\Theta}$ is active in the real dynamics. This guess is usually, for most columns, wrong. Because of this, many models must be created using different guess candidate functions. Furthermore, the hyperparameter λ_R used in the sequentially energy thresholded least-squares algorithm described in subsection 2.2.3. defines the sparsity of the solution and it also must be guessed. This means that for each candidate function guess, we must make another set of guesses of the hyperparameter value and create the respective models. The objective is to find the best model equation for each of the accelerations $\dot{x}_3$ and $\dot{x}_4$.

4.4. Picking the best candidate models

For each of the target variables, more than 200 models is generated. Many of these candidate models have non-sparse solutions ξ and can therefore be dropped immediately. The rest of the models must be evaluated according to some accuracy and simplicity metric.
For equal comparison, I reorder the models defined by the equation (15) after creating the model back to the implicit form, by moving the guess function θ_i back into the function library $\boldsymbol{\Theta}_i$ at the i-th column and adding an element into the i-th row of ξ_i with a value of -1. This

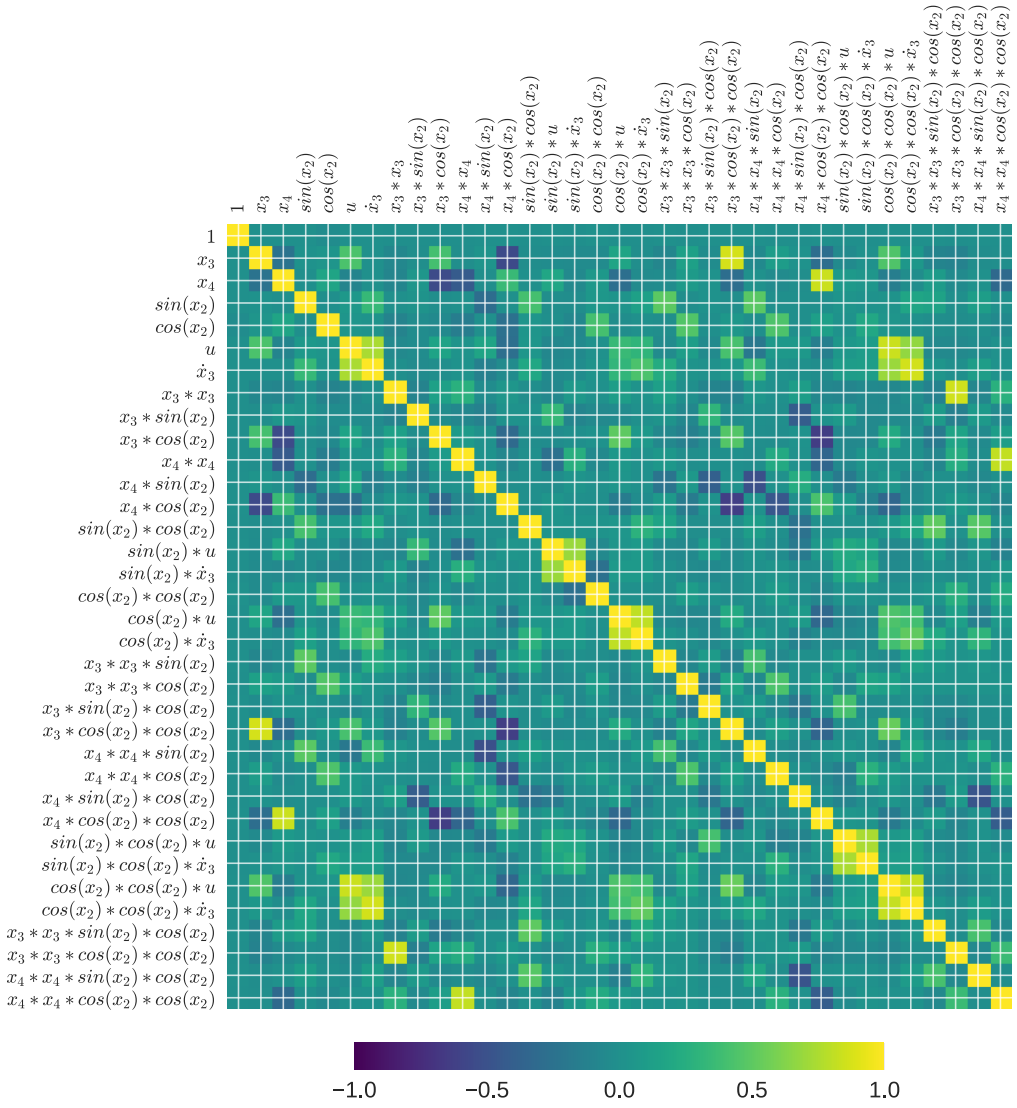
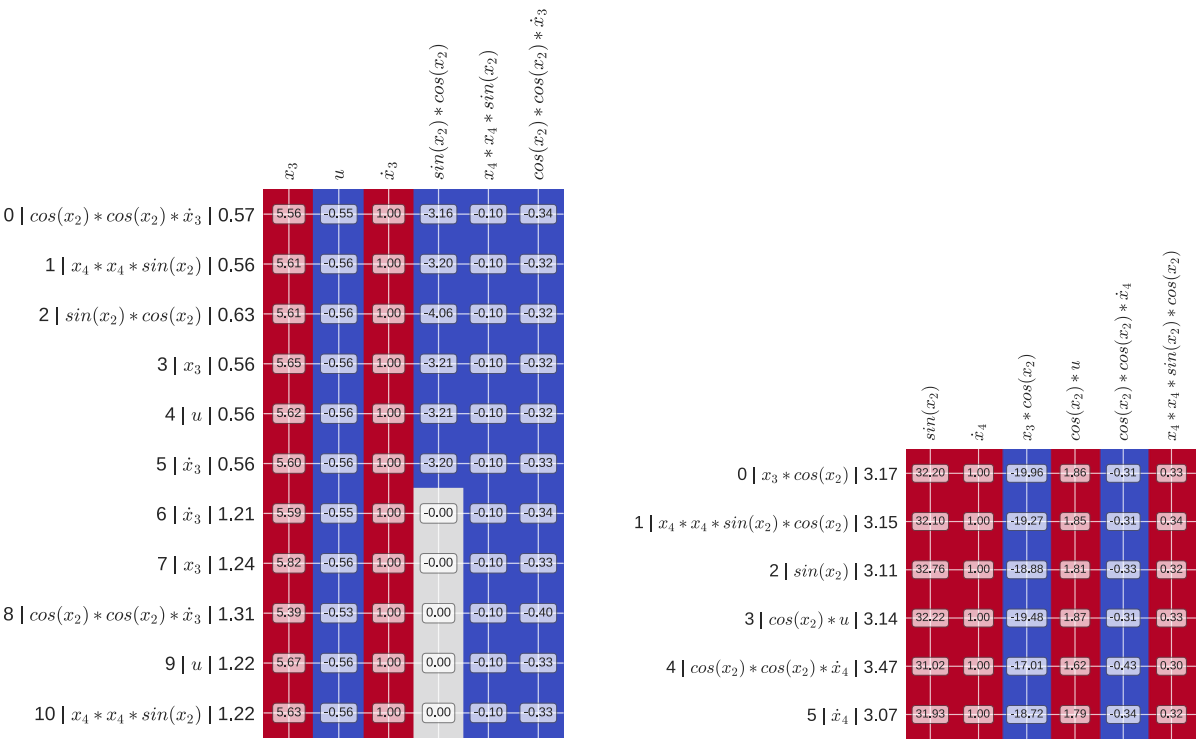


Fig. 4. *Correlation matrix for the function library used to generate models for $\dot{x}_3$. Shows the pair-wise correlation between candidate functions θ_i and θ_j, where i is the row index and j is the column index in the matrix.*

results in an implicit model $\boldsymbol{\Theta}(\mathbf{X}, \dot{\mathbf{X}})\boldsymbol{\xi} = \mathbf{0}$. Because the real model should be found whenever a correct candidate function guess θ_i is made, a model that appears consistently is likely to be correct. The solution vector $\boldsymbol{\xi}$ defining the models cannot however be compared directly, because there will always be a deviation in the parameters or the signs will be flipped. The objective is to find functions with the same active candidate functions. This can be done by picking the solutions $\boldsymbol{\xi}$ for each implicit model and creating another vector $\mathbf{a}$ whose elements have value 1 whenever the respective element in ξ_i is non-zero and 0 everywhere else. This vector $\mathbf{a}$, which I call the term activation vector, is computed for every candidate model. The L^1 distance in $\mathbf{a}$ between candidate models then simply says how many different active functions the models have, let's call this the *activation distance*. If two models have exactly the same active terms, then the activation distance is 0. After computing a distance activation matrix, which defines the activation distance between every pair of models, we can use clustering to find consistent models, which are likely to be the correct ones.

After clustering the models using their activation vectors, every model is given a cluster label. I only kept the models from clusters which contained 2 or more models. I then calculated the RMSE accuracy metric on those models. The implicit models for $\dot{x}_3$ and $\dot{x}_4$ are visualized in Figures 5. Each row represents a single model, with the respective y-axis tick describing the guess function θ_i which was used to generate the model and the model's training RMSE score. The columns represent the candidate functions. Candidate functions that didn't appear in any

of the models aren't visualized in the figures, because they'd make the plots too wide. The red and blue squares represent the signs of coefficients of $\boldsymbol{\xi}$ associated with the given candidate function, the square is red when the coefficient is positive, blue when it's negative and white when it's 0.



(a) *All unique implicit models for $\dot{x}_3$. trajectories.*

(b) *All unique implicit models for $\dot{x}_4$.*

Fig. 5. *The set of sparse, consistent and accurate identified implicit models. Each row corresponds to one model, with the y-axis labels specifying the model's index, guess function used to find the solution and its training RMSE metric.*

The implicit models are then solved for the respective accelerations $\dot{x}_3$ and $\dot{x}_4$, in this case using the Python's symbolic math package SymPy. The comparison of the reference model used for data generation and the best model identified from the data is in Table 2.

Table 2. *Comparison of the reference and the identified models.*

	Cart acceleration model $\dot{x}_1$ $\left[\frac{\mathrm{m}}{s^2}\right]$
Reference	$\frac{-0.66407u+6.64064x_3-0.11953x_4^2\sin(x_2)-0.02169x_4\cos(x_2)-1.91599\sin(2.0x_2)}{0.19531\cos(2.0x_2)-1.0}$
SINDy	$\frac{-0.55672u+5.61287x_3-0.10007x_4^2\sin(x_2)-1.61162\sin(2.0x_2)}{0.32312\cos^2(x_2)-1.0}$

	Pendulum angular acceleration model $\dot{x}_2$ $\left[\frac{\mathrm{rad}}{s^2}\right]$
Reference	$\frac{1.81554u\cos(x_2)-18.15533x_3\cos(x_2)+0.1634x_4^2\sin(2.0x_2)+0.18156x_4+32.05886\sin(x_2)}{0.3268\cos^2(x_2)-1.0}$
SINDy	$\frac{2.16696u\cos(x_2)-22.63092x_3\cos(x_2)+0.19338x_4^2\sin(2.0x_2)+38.39076\sin(x_2)}{0.19753\cos(2.0x_2)-1.0}$

4.5. Evaluating the acceleration models

Now that we have model equations for both accelerations $\dot{x}_3$ and $\dot{x}_4$, we can take a state measurement matrix $\mathbf{X}$ and external input measurement matrix $\mathbf{U}$, pass their rows $\mathbf{x}$ and u into each equation and get the predicted accelerations. The predicted accelerations can then be compared to the real accelerations as computed by the reference model.
The comparisons of derivative predictions are in Figures 6.

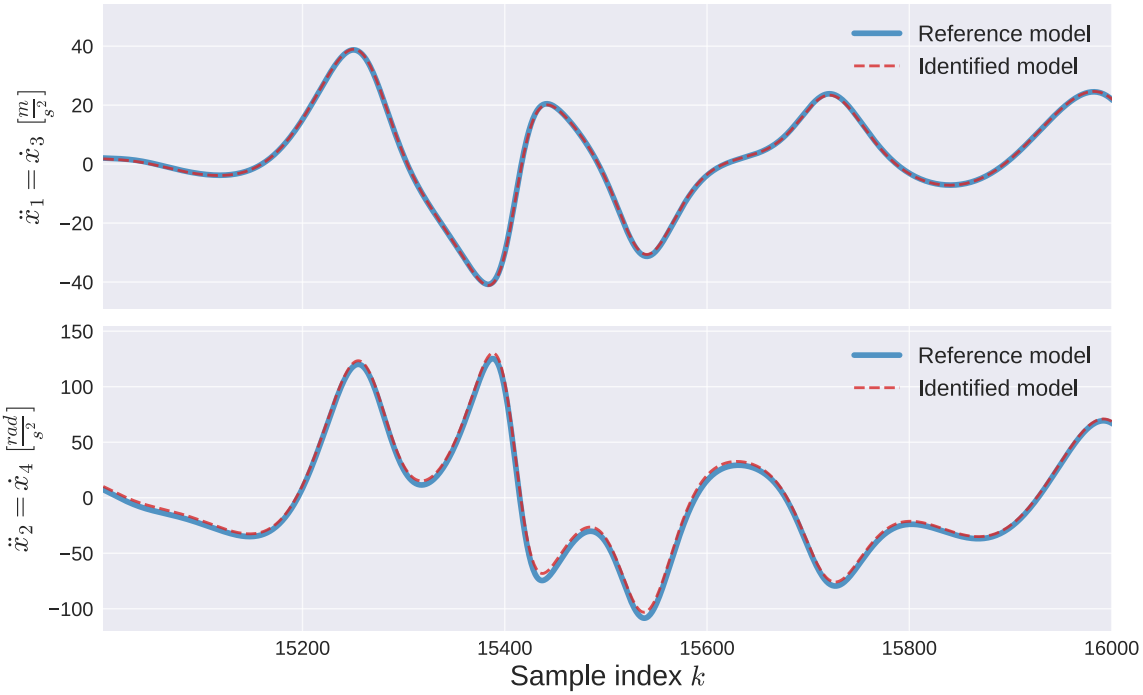


Fig. 6. *Comparison of real accelerations to accelerations computed from the state using the identified model.*

The model predictions are very close to the real accelerations, although there are some small deviations. Consolidating the acceleration models, $\dot{\mathbf{x}}$, into one state derivative vector equation creates a full state model. This way, the identified model can be simulated the same way as the analytically derived reference model. I simulated both from the same initial condition $\mathbf{x}[k = 0]$ and using the same input sequence $\mathbf{u}$. The trajectories for each state variable are visualized in Figure 7

The trajectories start out very close to each other, but small deviations eventually add up and the trajectories separate into relative chaos. In practice, the identified model would be used for its predictive capability. The accumulation of errors (differences between the model prediction and the real system) would be prevented with an estimator, for example a Kalman filter, that uses sensor feedback to determine the real state by weighing both the model prediction and sensor measurements.

5. Conclusion

An accurate nonlinear, rational model of the pendulum-cart system was identified from simulated measurement data with additive white noise. The data was first filtered by transforming the measurements into the frequency domain and setting the frequency components above the cutoff frequency to zero. This data was then numerically differentiated leveraging the properties of Fourier transforms to get the first and second derivatives needed for identification. Using the SINDy-PI method, an implicit ordinary differential equation was identified for each of the two accelerations in the system. From these two implicit models, the explicit equations were created by symbolically solving for the respective acceleration. The identified acceleration equations were accurate at predicting the acceleration given the state vector and external input value. By combining both acceleration equations, a full state model of the pendulum-cart system was created, which was then numerically simulated and compared to the real, analytically derived model used for generating the training data. The trajectories generated by both the real and identified model overlapped for the first few seconds, but the small errors between the models accumulated and the trajectories eventually decoupled. However, the models are nevertheless qualitatively nearly identical, and in practice, the imperfectness of the identified model would be largely compensated for via sensor feedback and state estimation.

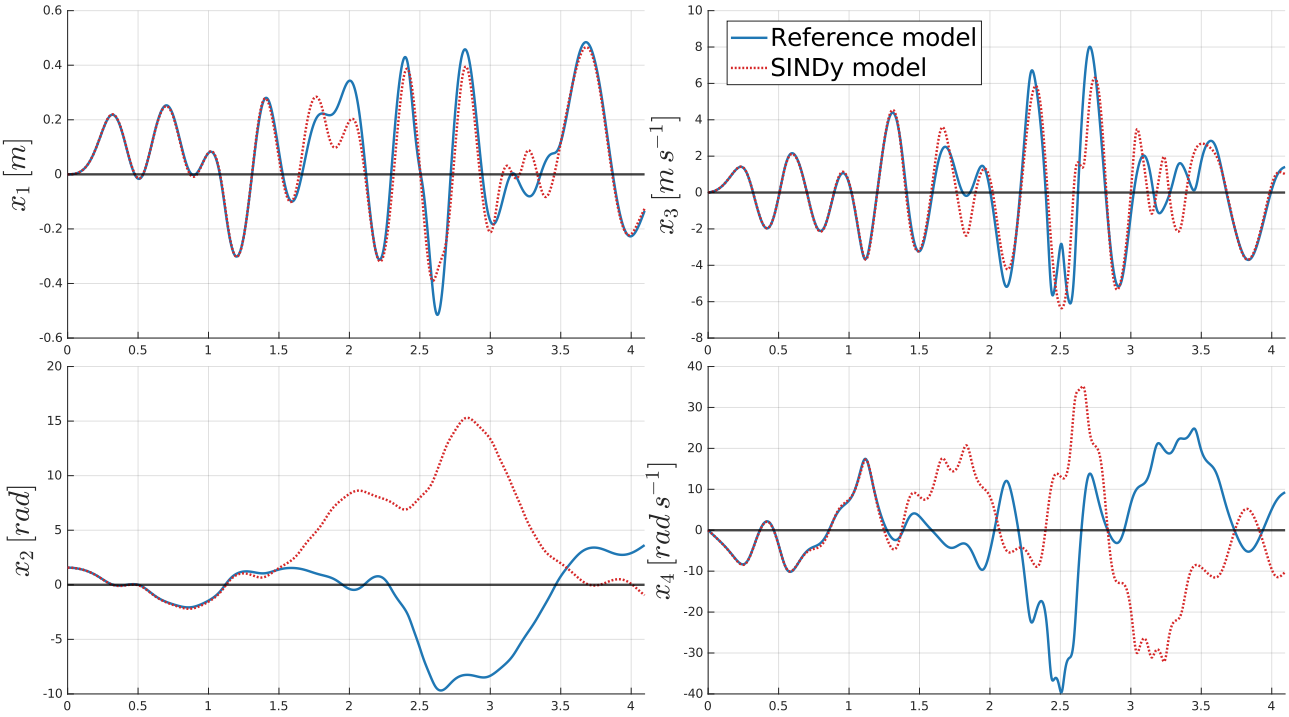


Fig. 7. *Parallel simulation of the reference model and the identified model. Full animation at: https://git.io/JBPXD*

Acknowledgement

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS19/158/OHK2/3T/12.

Nomenclature

x_1	Cart position (m)
x_2	Pendulum angle (rad)
$x_3, \dot{x}_1$	Cart velocity ($\mathrm{m\,s^{-1}}$)
$x_4, \dot{x}_2$	Pendulum angular velocity ($\mathrm{rad\,s^{-1}}$)
$\ddot{x}_1, \dot{x}_3$	Cart acceleration ($\mathrm{m\,s^{-2}}$)
$\ddot{x}_2, \dot{x}_4$	Pendulum angular acceleration ($\mathrm{rad\,s^{-2}}$)
$\mathbf{x}$	State vector ()
$\dot{\mathbf{x}}$	State derivative vector ()
$\boldsymbol{\Theta}$	Function library ()
$\boldsymbol{\xi}$	Vector of model parameters ()
$\mathbf{a}$	Activation vector ()
$\boldsymbol{X}$	Matrix of state measurements ()
$\dot{\boldsymbol{X}}$	Matrix of state derivative measurements ()
$\boldsymbol{U}$	Matrix of input measurements ()

References

[1] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Apr. 12, 2016). Publisher: National Academy of Sciences Section: Physical Sciences, pp. 3932–3937. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1517384113. Available at: https://www.pnas.org/content/113/15/3932 (visited on 03/31/2021).

[2] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Sparse Identification of Nonlinear Dynamics with Control (SINDYc)". In: *arXiv:1605.06682 [math]* (May 21, 2016). arXiv: 1605.06682. Available at: http://arxiv.org/abs/1605.06682 (visited on 03/31/2021).

[3] Niall M. Mangan et al. "Inferring biological networks by sparse identification of nonlinear dynamics". In: *arXiv:1605.08368 [math]* (May 26, 2016). arXiv: 1605.08368. Available at: http://arxiv.org/abs/1605.08368 (visited on 04/15/2021).

[4] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. Available at: https://doi.org/10.7717/peerj-cs.103.

[5] Kadierdan Kaheman, J. Nathan Kutz, and Steven L. Brunton. "SINDy-PI: A Robust Algorithm for Parallel Implicit Sparse Identification of Nonlinear Dynamics". In: *arXiv:2004.02322 [physics, stat]* (Sept. 29, 2020). arXiv: 2004.02322. Available at: http://arxiv.org/abs/2004.02322 (visited on 03/31/2021).

[6] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. 1st ed. Cambridge University Press, Jan. 31, 2019. ISBN: 978-1-108-38069-0 978-1-108-42209-3. DOI: 10.1017/9781108380690. Available at: https://www.cambridge.org/core/product/identifier/9781108380690/type/book (visited on 03/31/2021).

[7] Tobias Glück, Andreas Eder, and Andreas Kugi. "Swing-up control of a triple pendulum on a cart with experimental validation". In: *Automatica* 49.3 (Mar. 2013), pp. 801–808. ISSN: 00051098. DOI: 10.1016/j.automatica.2012.12.006. Available at: https://linkinghub.elsevier.com/retrieve/pii/S000510981200605X (visited on 04/11/2021).

Selected article from
Tento dokument byl publikován ve sborníku

Nové metody a postupy v oblasti přístrojové techniky, automatického řízení a informatiky 2021

New Methods and Practices in the Instrumentation, Automatic Control and Informatics 2021

15. 9. – 17. 9. 2021, Žatec

ISBN 978-80-01-06889-2

Web page of the original document:
http://iat.fs.cvut.cz/nmp/2021.pdf

Obsah čísla/individual articles:
http://iat.fs.cvut.cz/nmp/2021/

Ústav přístrojové a řídicí techniky, FS ČVUT v Praze, Technická 4, Praha 6