Application of sparse identification of nonlinear dynamics for physics-informed learning

Matteo Corbetta
SGT Inc., NASA Ames Research Center
Moffett Field, CA 94035
matteo.corbetta@nasa.gov

Abstract—**Advances in machine learning and deep neural networks has enabled complex engineering tasks like image recognition, anomaly detection, regression, and multi-objective optimization, to name but a few. The complexity of the algorithm architecture, e.g., the number of hidden layers in a deep neural network, typically grows with the complexity of the problems they are required to solve, leaving little room for interpreting (or explaining) the path that results in a specific solution. This drawback is particularly relevant for autonomous aerospace and aviation systems, where certifications require a complete understanding of the algorithm behavior in all possible scenarios. Including physics knowledge in such data-driven tools may improve the interpretability of the algorithms, thus enhancing model validation against events with low probability but relevant for system certification. Such events include, for example, spacecraft or aircraft sub-system failures, for which data may not be available in the training phase. This paper investigates a recent physics-informed learning algorithm for identification of system dynamics, and shows how the governing equations of a system can be extracted from data using sparse regression. The learned relationships can be utilized as a surrogate model which, unlike typical data-driven surrogate models, relies on the learned underlying dynamics of the system rather than large number of fitting parameters. The work shows that the algorithm can reconstruct the differential equations underlying the observed dynamics using a single trajectory when no uncertainty is involved. However, the training set size must increase when dealing with stochastic systems, e.g., nonlinear dynamics with random initial conditions.**

TABLE OF CONTENTS

1. INTRODUCTION

Machine learning is revolutionizing several branches of science and engineering. It has enhanced the capability to analyze and interpret data [1], perceive external environment through sensors, image processing [2], enable compressed representations of complex systems [3], and many others. As tasks became more complex, so the complexity of the machine learning solutions utilized to accomplish them. The results are complex algorithms capable of grinding large amount of data and producing outstanding results. This comes with costs. First, the obvious need of data. Large amount of data to train deep networks are not always available. In the optimal case, training sets should span the domain expected to be observed after the algorithm deployment, but this condition too may be prohibitive for a variety of scenarios. Second, the complexity of the tasks drive up the number of parameters to be learned during training, which means risk of overfitting and reduced generalization, thus requiring even more data to avoid the latter. Then, the lack of interpretability of such complex algorithms has raised concerns in recent years. Those concerns are driven by the need to trust a machine decision, and even more importantly, to understand the reasoning behind it [4]. Consequently, the risk-adverse aerospace and aeronautical domains are slower than others in embracing advanced machine learning and artificial intelligence. Systems have to be certified by regulators, and besides some recent efforts [5], methodologies have not been consolidated yet. The algorithms should be capable of handling scenarios not observed during the training phase, for example, sensor signals representative of sub-system failures. In addition, it is unclear how to handle the stochastic or probabilistic nature of some algorithms. Those produces different results at different runs because the solutions reflect a local minimum of the objective function, thus hindering deployment in safety-critical areas.

The need for interpretable algorithms lead torwards new research areas like *explainable* AI [4], interpretable machine learning [6], as well as physics-informed learning. Some recent works on physics-informed learning attempted at solving physics problems, described by sets of differential equations, by adopting machine learning algorithms. In those cases, the cost functions were opportunely modified by introducing physical constraints and symmetries that lead to solutions conforming to the underlying phenomenon [7] [8]. Other works focused on the discovery of governing equations of dynamical systems from data [9], which has some potential advantages. It would intrinsically enhance extrapolation and generalization. If an algorithm were capable of identifying the links between the observed quantities through physical relationships, it could be applied to other specimens of the same system with no or minimal tuning. Physical relationships would also help extrapolating the system behavior outside of the training domain. Aerospace and aeronautics would benefit of this last property, since a variety of scenarios, especially related to anomalies, component and sub-system failures, may have no data available for training and/or validation.

This paper was inspired by the work of Brunton et al. [9], where sparse regression was proposed as a tool to learn governing equations of dynamical systems from data. This paper explores the approach by applying it to two dynamical systems, a modified *Van der Pol oscillator*, where a nonlinearity was added in the zero-order term (i.e., stiffness term), and a first order, three-dimensional system with random initial conditions on two of the three dimensions. The paper shows the application of the algorithm to those two case studies, with some qualitative but critical analysis. Sparse regression appears to require very limited amount of data when compared to typical machine learning methods. However, the 3D stochastic system required considerably mode data than the modified Van der Pol oscillator to learn the relationships between the quantities of interest. Finally the paper discusses some aspects related to the application and deployment on real systems.

2. Summary of sparse regression for system identification

This section summarizes the algorithm utilized in this work, originally proposed in [9]. The notation utilized in this section and throughout the manuscript is the following. Lowercase letters, e.g., x, indicate scalar quantities and bold lowercase letters, e.g., $\boldsymbol{x}$, indicates vectors. Matrices are defined through bold, capital letters, e.g., $\boldsymbol{X}$, while parenthesis stress the dependency in functions and vector functions, e.g., $\boldsymbol{f}(\cdot)$. Time dependencies, e.g., $\boldsymbol{x}(t)$, are emphasized when necessary.

The algorithm proposed in [9], called sparse identification of nonlinear dynamics (SINDy), aims at estimating the structure of a potentially highly-nonlinear differential equation,

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left(\boldsymbol{x}(t)\right) \quad ,$$

where $\boldsymbol{x}(t) \in \mathbb{R}^{n\times 1}$ is the *n*-dimensional state vector at time t, and $\boldsymbol{f}(\cdot) : \mathbb{R}^{n\times 1} \to \mathbb{R}^{n\times 1}$ is a *n*-dimensional state mapping function. The goal is to estimate the actual form of the function $\boldsymbol{f}(\cdot)$, using data.

As the authors stressed in the original paper, the assumption allowing the algorithm to converge to the solution is that only a few elements compose function $\boldsymbol{f}(\cdot)$, making it sparse in the space of possible functions [9]. Sparsity balances model complexity and accuracy, and the method weights the regression by the number of non-zero elements. The goal of SINDy can be defined through a ℓ_1-regularized regression:

$$\boldsymbol{\xi}_k = \arg\min_{\hat{\boldsymbol{\xi}}_k} ||\dot{\boldsymbol{X}} - \hat{\boldsymbol{\xi}}_k \boldsymbol{\Theta}\left(\boldsymbol{X}\right)||_2 + \lambda\, ||\hat{\boldsymbol{\xi}}_k||_1 \quad , \qquad (1)$$

where all elements are described hereafter.

Matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ is composed of m snapshots of the state vector at different time steps, as in Eq. (2), where subscripts $1, 2, \cdots, n$ indicates the elements within the *n*-dimensional state vector, and t_i indicates the *i*-th time step.

$$\boldsymbol{X}(t) = \overset{\xrightarrow{\text{state dimension}}}{\begin{bmatrix} | & | & \cdots & | \\ x_1(t_i) & x_2(t_i) & \cdots & x_n(t_i) \\ | & | & \cdots & | \end{bmatrix}} \Big\downarrow \text{time} \qquad (2)$$

The snapshot of the state derivatives, $\dot{\boldsymbol{X}}$, follows the same structure. It can be approximated by differentiation of the elements in $\boldsymbol{X}$, although some filtering may be required to remove noise-driven oscillations [9]. Matrix $\boldsymbol{\Theta}(\boldsymbol{X})$ represents one of the key novelties of the method. It is a *candidate function library*, where each column represents a potential candidate for the elements in $\boldsymbol{f}(\cdot)$ to be discovered. The selection of functions to populate the library is arbitrary and may be composed of polynomial terms, trigonometric functions, etc., [9], Eq. (3).

$$\boldsymbol{\Theta}\left(\boldsymbol{X}\right) = \begin{bmatrix} | & | & | & \cdots & | & | & \cdots \\ \mathbf{1} & \boldsymbol{X} & \boldsymbol{X}^{p_2} & \cdots & \sin \boldsymbol{X} & \cos \boldsymbol{X} & \cdots \\ | & | & | & \cdots & | & | & \cdots \end{bmatrix} \qquad (3)$$

Matrix $\boldsymbol{\Theta}(\boldsymbol{X})$ is constructed by stacking together, column by column, candidate nonlinear functions of $\boldsymbol{X}$. For example, element $\mathbf{1}$ is a column-vector of ones, element $\boldsymbol{X}$ is already defined in (2), element $\boldsymbol{X}^{p_2}$ is the matrix containing the set of all quadratic polynomial functions of the state vector $\boldsymbol{x}$, and is constructed as follows:

$$\boldsymbol{X}^{p_2}(t) = \begin{bmatrix} x_1^2(t_1) & x_1\, x_2(t_1) & \dots & x_2^2(t_1) & \cdots & x_n^2(t_1) \\ x_1^2(t_2) & x_1\, x_2(t_2) & \dots & x_2^2(t_2) & \cdots & x_n^2(t_2) \\ \vdots & \vdots & \dots & \vdots & & \\ x_1^2(t_m) & x_1\, x_2(t_m) & \dots & x_2^2(t_m) & \cdots & x_n^2(t_m) \end{bmatrix} .$$

The superscript p_2 has been used to define the set of quadratic polynomial functions, and should not be confused with, for example, "x to the power of p_2". Similarly, $\boldsymbol{X}^{p_3}$ defines the set of cubic polynomial functions, and so on.

Vector $\boldsymbol{\xi}$ collects the coefficients of the candidate functions in $\boldsymbol{\Theta}(\boldsymbol{X})$, and is the objective of the minimization. The number of vectors equals the dimension of the state vector, n, so $k = 1, \ldots, n$. Since only a few of the candidate functions are expected to have an effect on the system dynamics, all vectors $\boldsymbol{\xi}$ are expected to be sparse. Symbols $||\cdot||_2$ and $||\cdot||_1$ indicates *norm*-2 and *norm*-1, respectively, while λ is a scalar multiplier. Element $\lambda\, ||\hat{\boldsymbol{\xi}}_k||_1$ is the regularization term, which penalizes coefficients different from 0 in a linear fashion. It is the actual promoter of sparsity in the minimization problem.

Once all sparse vectors $\boldsymbol{\xi}_k$ have been estimated (see following subsection), they can be collected in the sparse matrix $\boldsymbol{\Xi}$,

$$\boldsymbol{\Xi} = \begin{bmatrix} | & | & \cdots & | \\ \boldsymbol{\xi}_1 & \boldsymbol{\xi}_2 & \cdots & \boldsymbol{\xi}_n \\ | & | & \cdots & | \end{bmatrix} \quad ,$$

so that the system dynamics can be computed as in Eq. (4).

$$\dot{\boldsymbol{X}}(t) = \boldsymbol{\Theta}(\boldsymbol{X}(t))\boldsymbol{\Xi} \qquad (4)$$

Solution of sparse regression

Equation (1) poses the estimation problem in the same terms of the LASSO (*least absolute shrinkage and selection operator*), [10]. However, [9] suggests the use of *sequential thresholded least square*, which will be also used to produce the graphs in this paper. While the LASSO solution requires convex optimization algorithms or the solution by least angle regression [11], sequential thresholded least square imposes sparsity by "manually" setting all coefficients smaller than λ to 0 in an iterative fashion. The two methods produced very similar results in many of the tested cases. However, the results from the LASSO were not reported for the sake of brevity, since do not add intuition to the discussion. The sequential thresholded least square from [9] is summarized by the Python code in Table 1[2]. The result is the sparse

[2] A Matlab version of the code can be found in the supporting material of the original paper from Brunton et al.

Table 1. Sequential thresholded least square in Python. The linear equation is solved using the QR decomposition.

```
import numpy as np

def linearEqSolver(A, b):
    Q, R = np.linalg.qr(A)
    p = np.dot(Q.T, b)
    return np.dot( np.linalg.inv(R), p )

Xi = linearEqSolver(Theta, dxdt)                # Initialize Xi values
for iteration in range(niter):
    smallCoeff = abs(Xi) < lmbd                 # get coeff < lambda
    Xi[smallCoeff] = 0.0                        # set them to 0
    for dim in range(ndim):                     # regress over large coeffs
        bigCoeff = np.logical_not(smallCoeff[:, dim]).reshape((-1,))
        Theta_tmp = Theta[:, bigCoeff]
        dxdt_tmp = dxdt[:, dim]
        Xi[bigCoeff, dim] = linearEqSolver(Theta_tmp, dxdt_tmp)
```

matrix $\boldsymbol{\Xi}$ collecting the coefficients of the candidate functions in $\boldsymbol{\Theta}(\boldsymbol{X})$. If, as expected, the dynamic of the system is sparse with respect to the space of possible functions, most of the elements in $\boldsymbol{\Xi}$ will be zero. Otherwise, $\boldsymbol{\Xi}$ will be a full matrix, suggesting the solution is not sparse.

The number of iterations (*niter* in Table 1) is not defined here. Several trials showed that a few iterations (from a couple up to 5 or 10) were sufficient to estimate the dynamics of the two systems presented in Section 3. Performance appears independent on the number of iterations in those test cases. If the algorithm was not capable of identifying the correct dynamics after a few iterations, then it never converged. Adding more iterations did not improve the selection capability. This behavior may not hold for larger systems (large n).

3. APPLICATION

This section presents the application of the SINDy algorithm to two nonlinear systems. All results and graphics reproduced here were generated using Python 3.6 [12], libraries *NumPy* and *SciPy* [13], and *matplotlib* [14].

Modified Van der Pol Oscillator

This subsection shows the application of SINDy to the Van der Pol oscillator equation [15], modified by an extra nonlinear term and perturbed by sinusoidal excitation. The nonlinear dynamics is defined by the scalar second-order differential equation:

$$\ddot{x} - \frac{c}{m}(1-x^2)\dot{x} + x\left(\frac{k}{m} + \frac{k_{nl}}{m}x^2\right) = \frac{1}{m}F(t) \quad . \qquad (5)$$

The element $(1-x^2)$ in the second term on the left-hand side distinguish the Van der Pol oscillator from the standard linear oscillator, and the term $k_{nl}\,x^2$ has been added here to introduce further non-linearity. Mass-normalized parameters c/m, k/m, and k_{nl}/m are positive scalar quantities, while $F(t)$ is the sinusoidal excitation. The values utilized in this simulation are reported in Table 2. Since Eq. (5) is a second-order differential equation, it has been transformed into a state-space formulation, i.e., $\boldsymbol{x} = [x, \dot{x}]^T$. The dynamic was integrated using Euler's forward method with time step size $\Delta t = 1\mathrm{e}-3\,\mathrm{s}$, and results were used as input data for SINDy. The acceleration was corrupted by Gaussian noise with standard deviation $\sigma = 2.0$ to replicate possible noise from acceleration measurements, Figure 1.

Table 2. Parameters of the modified Van der Pol oscillator and initial conditions for the simulation.

Parameters	**Values**
c/m	0.16
k/m	0.25
k_{nl}/m	1.25
$F(t) = F_0 \sin(2\pi f t + \phi)$	$F_0 = 200$ N $f = 0.2$ Hz $\phi = 0$ rad
Initial conditions	
$x(t=0)$	2.75 m
$\dot{x}(t=0)$	0 m/s

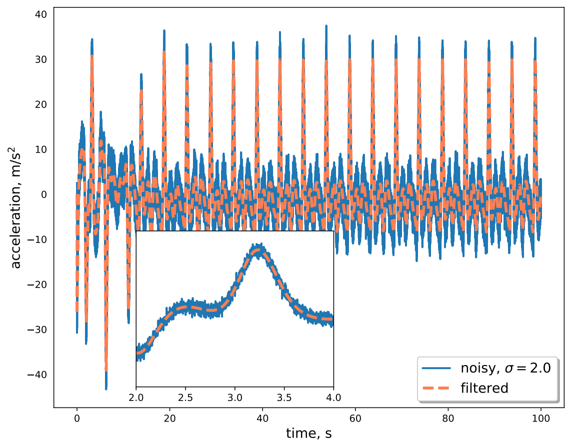

Figure 1. Acceleration signal corrupted by $r \sim \mathcal{N}(0,2)$ and filtered.

In this mechanical case study, the solution through SINDy requires availability of position, velocity and acceleration. The full kinematic profile may not be available in experimental settings where kinematics is measured through sensors. Numerical differentiation and subsequent filtering may help overcoming the problem, as already suggested in [9]. The use of two sensors, e.g., displacement transducers and accelerometers, could potentially help the algorithm, thus requiring only one numerical differentiation.

Figure 2 shows position and velocity of the modified forced Van der Pol oscillator in time domain. A phase plot is also shown to appreciate the nonlinear dynamics. The 3D plot shows the state variables as time passes by, where the first half of the simulation was used as training data (blue line). The dynamics learned from the SINDy algorithm (dashed green line) is then compared against the second half of the simulation (orange line) for validation.

The dynamics learned through SINDy is reported in Table

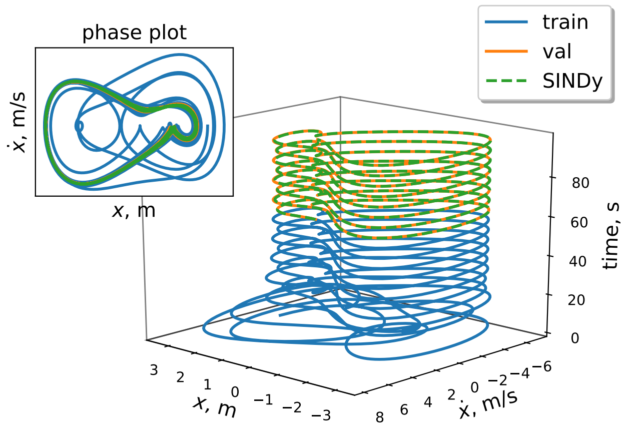

Figure 2. Dynamics of the modified forced Van der Pol oscillator learned through SINDy.

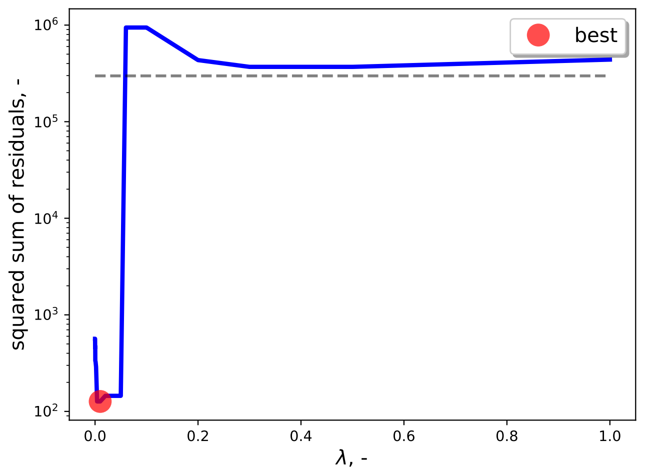

Figure 3. Selection of regularized parameters through AIC based on squared sum of residuals. The dashed-gray line represents the error in the limit case $\xi = 0$, attained when λ is too high.

Table 3. Sparse vector ξ_2 from SINDy.

Θ-column	ξ_2	Reference	Error, %
1	0.0	0	-
x	-0.24938066	-0.25	0.25
$\dot{x}$	0.15618899	0.16	2.38
x^2	0.0	0.0	-
$x\dot{x}$	0.0	0.0	-
$\dot{x}^2$	0.0	0.0	-
x^3	-1.25049414	-1.25	0.04
$x^2\dot{x}$	-0.15509556	-0.16	3.0
$x\dot{x}^2$	0.0	0.0	-
$\dot{x}^3$	0.0	0.0	-
$F(t)$	0.05003716	0.05	0.07

3 (vector $\boldsymbol{\xi}_1$, because of the state-space formulation, is all 0 except for the velocity term $\dot{x}$ equal to 1). The candidate function library was built using sets of polynomials up to the third order, so using the matrix $\boldsymbol{\Theta}(\boldsymbol{X})$ presented in Eq. (3) up to the element $\boldsymbol{X}^{p_3}$, and adding, as last column, the forcing $F(t)$. Similar results were obtained by increasing the size of the candidate function library as well as adding trigonometric functions; all estimates of the non-necessary coefficients were null. Adding the external forcing is fundamental. If $F(t)$ were not included, then SINDy would try to link, erroneously, the effect of $F(t)$ to the other columns in $\boldsymbol{\Theta}$.

The regularization parameter λ is the sparsity promoter. Small λ would cause the resulting matrix $\boldsymbol{\Xi}$ not to be sparse, and large λ would force excessive sparsity, with the limit case $\boldsymbol{\xi}_k = \boldsymbol{0}\ \forall\ k = 1, ..., n$. A (sub-)optimal λ can be selected in a typical machine learning fashion, by splitting the available kinematic profile in training and validation. First, select an initial λ. A segment of the kinematic profile is then used to train the algorithm by imposing the selected λ as regularizer, and the remaining segment of the kinematic profile is simulated through SINDy, Eq. (4). The error between the SINDy-replicated kinematics and validation data is then used as metric to assess the goodness of the selected λ. The procedure can be repeated sweeping through a set of possible λ values. In the proposed example, 28 values between 0 and 1 were used to define the set of possible λ, [0, 1e-5, 2e-5, ..., 0.001, 0.002, ..., 0.1, 0.2, ..., 1]. The best λ was selected using the Akaike information criterion (AIC) computed over the squared sum of residuals, as in [16]:

$$\epsilon = \sum_{i=1}^{m} \left(x(t_i) - x_{\text{SINDy}}(t_i)\right)^2$$
$$\text{AIC}_\gamma = m \log \frac{\epsilon}{m} + 2\gamma \quad ,$$

where m is the number of datapoints in the validation set (similarly to the notation in the previous section, m represents the number of data points over time) and γ is the number of non-zero parameters in the SINDy model. In this instance, the squared sum of residuals was computed using the position profile only. The resulting AIC_γ is the value corresponding to γ non-zero parameters, and the model with the lowest AIC is selected as the best performing. Figure 3 shows the resulting sum of squared residuals and the best regularization value chosen with AIC, $\lambda = 0.02$ (the results shown in Fig. 2 were generated with such λ).

Nonlinear system with random initial conditions

This subsection shows the application of SINDy to the stochastic system:

$$\begin{aligned} \frac{\mathrm{d}y_1}{\mathrm{d}t} &= y_1 y_3, \\ \frac{\mathrm{d}y_2}{\mathrm{d}t} &= -y_2 y_3, \\ \frac{\mathrm{d}y_3}{\mathrm{d}t} &= -y_1^2 + y_2^2. \end{aligned} \tag{6}$$

The initial conditions are: $y_1(0) = 1$, $y_2(0) = 0.1\,u$, and $y_3(0) = 0$, where $u \sim \mathcal{U}[-1, 1]$. The system was first utilized in [17] to test deep-residual recurrent neural networks. SINDy is not capable of learning the correct dynamics using a single trajectory because of the random initial condition of y_2. The result is intuitive, since the random initial condition drives trajectory y_2 on either positive or negative side of the set of real numbers and that changes at every run (see Fig. 4 discussed below). The algorithm is obviously incapable of understanding the true dynamic with a single trajectory. The addition of several trajectories to the dataset helps identifying the correct sparse coefficients. The latter can be performed by simply row-stacking all trajectories together. Therefore, the number of rows of matrix $\boldsymbol{X}$ (Eq. (2)) will increase with the number of sample trajectories. In this example, $n = 3$ and the number of sample points in time domain is m, so $\boldsymbol{X} \in \mathbb{R}^{m \times 3}$. By stacking T trajectories together, the dimension of $\boldsymbol{X}$ becomes $Tm \times 3$. All other dimensions of the matrices required by SINDy will follow accordingly.

To test the effect of randomness on the system identification capabilities, a number of trajectories $\boldsymbol{y}(t)$ were simulated for 20 seconds using $\Delta t = 0.05$ s. The derivatives $\mathrm{d}\boldsymbol{y}/\mathrm{d}t$ were computed by numerical differentiation and adding Gaussian noise with $\sigma = 0.2$. Figure 4 and 5 show the system simulation unrolled in time domain, and a 3D plot, where time is represented by colors (from blue to red). If only one of those trajectories were fed into SINDy, the resulting vectors $\boldsymbol{\xi}_k$ were not sparse. Several values of λ, ranging from $1e-3$ up to the order of $1e3$, were tested in a trial and error fashion, but none of them produced reasonably accurate results. Instead, by using multiple sample trajectories, the correct sparse vectors were identified even by varying λ. Figure 6 shows the result after training SINDy using 200 sample trajectories and $\lambda = 0.5$ (dot-dashed red line), against the results after training with a single trajectory (dashed orange line, also with $\lambda = 0.5$). The thick dark gray curve was left out of the training and used as validation. The structure of the function learned with a single curve produces a trajectory that becomes unstable after a couple of iterations, falling outside of the range of values, while adding multiple sample paths during training allows SINDy to successfully reconstruct Eq. (6) with the following non-zero elements:

$$\begin{aligned} y_1\,y_3 &= 0.996069 \quad , \\ y_2\,y_3 &= -0.994354 \quad , \\ y_1^2 &= -0.997278\,, y_2^2 = 0.99549 \quad , \end{aligned}$$

where each row refers to the first, second and third equation, respectively.

The introduction of multiple trajectories in the dataset appears to be robust to more randomness. The original initial conditions were modified by adding $y_3(0) = u$, where u is again a random realization from the uniform distribution with range [-1,1] (different from the realization for $y_2(0)$). Figure 7 shows the system dynamics unrolled in time, while Figure 8 shows a 3D representation. In this case, the number of sample trajectories used for training had to be increased to 400 to correctly capture the system dynamic and so to provide accurate sparse vectors $\boldsymbol{\xi}$. The regularization parameter λ was kept to 0.5. Nonetheless, the estimation of a (sub-)optimal λ as done in the previous example would help the regression problem and a lower number of trajectories may be required. Figure 9 shows the SINDy results using 400 samples, compared against a reference validation curve and the results using a single curve for training. The outcome corroborates what was observed in the previous case study with one random initial condition only.

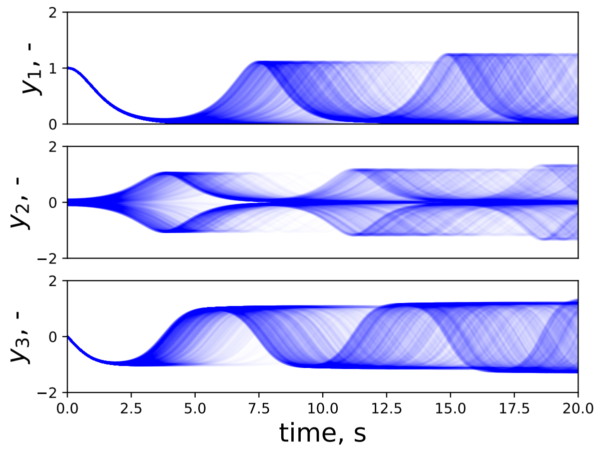

Figure 4. 1500 sample trajectories $y(t)$ unrolled in time domain. For clarity, only a subset of all the trajectories has been shown.

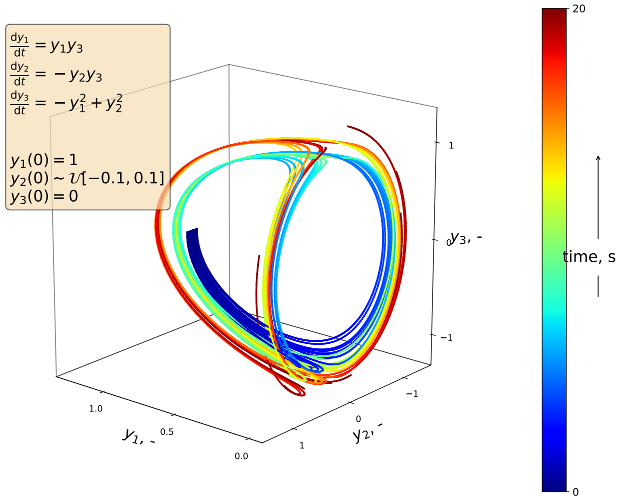

Figure 5. 3D representation of the system dynamics. For clarity, only a subset of all the trajectories has been shown.

4. CONCLUSIONS

This work proposed a qualitative analysis of the SINDy algorithms in two test scenarios, where nonlinear dynamical systems served as reference. From the short analysis presented here, a number of advantages and drawbacks arise from the use of SINDy.

Sparsity enables simultaneous model selection and parameter identification, which helps generating parsimonious models (with low number of parameters and functions) when the true dynamics of the system is unknown. SINDy produces models based on candidate functions of the state variables, so it paves the way to easy interpretation, and may also aid causal inference. This is a clear advantage when comparing SINDy against other machine learning techniques, were the parameters utilized for regression lose (completely or partially) physical meaning. The computational cost appears to be low,

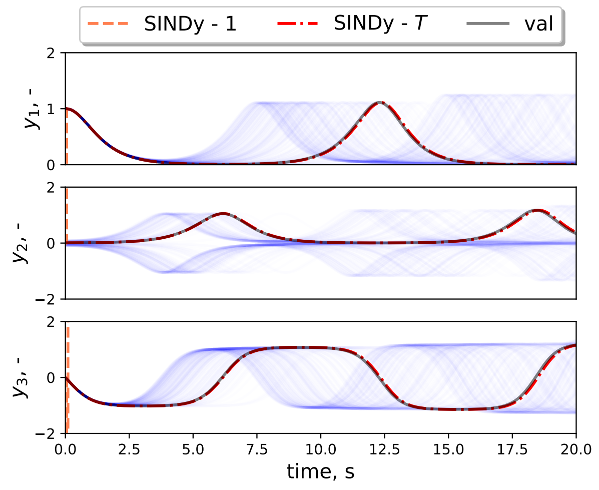

Figure 6. **System identification using a single curve and 200 random trajectories. For clarity, only a subset of all the trajectories has been shown.**

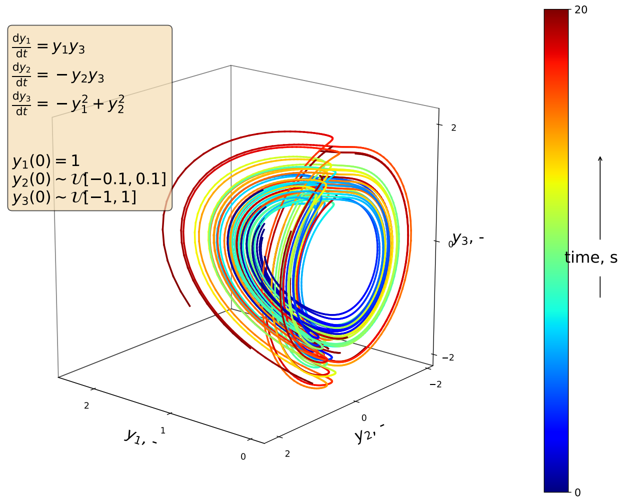

Figure 8. **3D representation of system dynamics with the additional random initial condition $y_3(0) = u$. For clarity, only a subset of all the trajectories has been shown.**

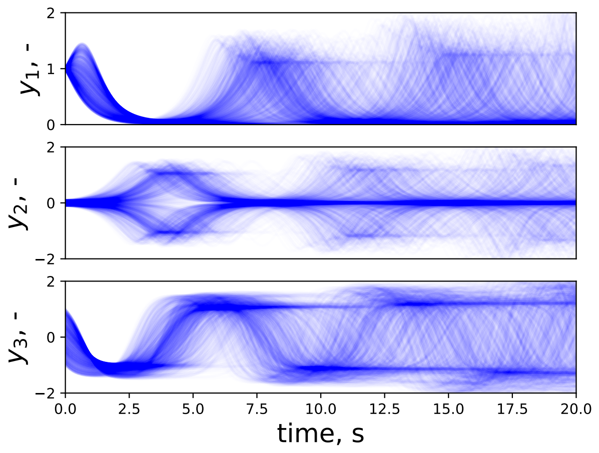

Figure 7. **1500 sample trajectories $y(t)$ unrolled in time domain, with the additional random initial condition $y_3(0) = u$. For clarity, only a subset of all the trajectories has been shown.**

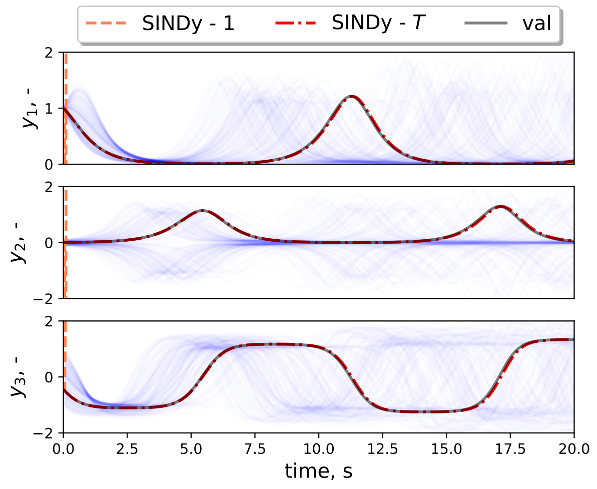

Figure 9. **System identification with $y_3(0) = u$, using a single curve and 400 random trajectories. For clarity, only a subset of all the trajectories has been shown.**

at least for the low-dimension problems shown in this paper. Moreover, the algorithm allows the straightforward stacking of multiple sample trajectories in the dataset of snapshots $\boldsymbol{X}$, thus enhancing the regression capability for stochastic systems.

A number of issues emerged by the implementation of the algorithm. The primary concern is the candidate functions library $\boldsymbol{\Theta}(\boldsymbol{X})$. If one of the snapshots of the true functions is missing from the library, then the algorithm will try to associate the effect of that missing term to the columns in $\boldsymbol{\Theta}(\boldsymbol{X})$. As a result, the coefficient vectors $\boldsymbol{\xi}_k$ might be incorrect or not sparse, failing to identify the correct dynamics. In those cases, the extrapolation capabilities of SINDy using Eq. (4) can be extremely poor. Further improvements of the methodology are likely to be required for real applications, because when the true dynamic of the system is unknown, it is not efficient, nor sound, to select a huge set of candidate functions in the library hoping to include the correct one. Such a problem does not apply to the simple cases shown here, where the dimension of the state variable was either 2 or 3 and the elements constituting the differential equations were fairly simple.

Systems subjects to exogenous forces may be identified easily, provided that external forcing is measured and embedded in the candidate function library. On the other hand, systems with feedback correction loops, e.g., $K(\boldsymbol{x} - \boldsymbol{x}_{\text{des}})$, pose a problem. If the control system is simple enough that feedback correction can be split, and $K\boldsymbol{x}$ and $K(\boldsymbol{x}_{\text{des}})$ can be embedded in the function library (for example in proportional or proportional-derivative control), then SINDy still works.

Otherwise, it would become impossible to disambiguate the effect of the feedback control from the un-controlled system dynamic [9].

The outcome from this qualitative analysis may change when testing SINDy in high-dimensional spaces. If the system dynamic is described by a combination of non-rational and rational functions, the sequential thresholded least square should be modified by first constructing the library $\boldsymbol{\Theta}(\boldsymbol{X})$ in a specific fashion, and then compute the null space of the new library, as proposed in [18]. The sampling frequency has not been discussed here, but it can also pose challenges. If measurements of the system state variables were coarse, i.e. low sampling rate, the algorithm might not converge to the correct sparse vectors.

Uncertainty quantification is also relevant for this type of algorithms, and Bayesian alternative could be explored. As a matter of fact, Park and Casella proposed, in 2008, the Bayesian LASSO, [19], which is based on Gibbs sampling and could be implemented to solve Eq. (1). However, several attempts have been carried out to apply the Bayesian LASSO to the case studies proposed here, and none of them was successful. Specifically, the target non-zero elements were underestimated (closer to 0 than they should be), and the target zero elements were overestimated (larger than 0). Although the cause of those unsuccessful results was not identified, the use of Bayesian methods in sparse regression problems is still a matter of discussion.

Acknowledgments

This work was supported by the System-Wide Safety (SWS) project under the Airspace Operations and Safety Program within the NASA Aeronautics Research Mission Directorate (ARMD).

References

[1] S. Ekins, A. C. Puhl, K. M. Zorn, T. R. Lane, D. P. Russo, J. J. Klein, A. J. Hickey, and A. M. Clark, “Exploiting machine learning for end-to-end drug discovery and development,” *Nature materials*, vol. 18, no. 5, p. 435, 2019.

[2] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in *Advances in neural information processing systems*, 2017, pp. 3856–3866.

[3] S. L. Brunton and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[4] D. Gunning, “Explainable artificial intelligence (xai),” *Defense Advanced Research Projects Agency (DARPA), nd Web*, vol. 2, 2017.

[5] C. Wilkinson, J. Lynch, R. Bharadwaj, and K. Woodham, “Verification of adaptive systems,” 2016, fAA Technical Report.

[6] F. Doshi-Velez and B. Kim, “Towards a rigorous science of interpretable machine learning,” *arXiv preprint arXiv:1702.08608*, 2017.

[7] A. Karpatne, W. Watkins, J. Read, and V. Kumar, “Physics-guided neural networks (pgnn): An application in lake temperature modeling,” *arXiv preprint arXiv:1710.11431*, 2017.

[8] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations,” *arXiv preprint arXiv:1711.10561*, 2017.

[9] 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.

[10] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[11] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, “Least angle regression,” *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[12] G. v. Rossum, “Python tutorial, technical report cs-r9526,” in *Centrum voor Wiskunde en Informatica (CWI), Amsterdam*, 1995.

[13] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001. [Online]. Available: ”http://www.scipy.org/”

[14] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[15] B. Van der Pol, “A theory of the amplitude of free and forced triode vibrations, radio rev. 1 (1920) 701-710, 754-762; selected scientific papers, vol. i,” 1960.

[16] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor, “Model selection for dynamical systems via sparse regression and information criteria,” *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, no. 2204, p. 20170009, 2017.

[17] J. N. Kani and A. H. Elsheikh, “Dr-rnn: A deep residual recurrent neural network for model reduction,” *arXiv preprint arXiv:1709.00939*, 2017.

[18] 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*, vol. 2, no. 1, pp. 52–63, 2016.

[19] T. Park and G. Casella, “The bayesian lasso,” *Journal of the American Statistical Association*, vol. 103, no. 482, pp. 681–686, 2008.

Biography

Matteo Corbetta *is a Research Engineer with SGT Inc. at NASA Ames Research Center, CA, where he is investigating uncertainty quantification methods, model-based and data-driven algorithms for diagnostics and prognostics applied to autonomous systems. Prior to joining NASA, he worked as R&D Condition Monitoring Systems Engineer at Siemens Wind Power, Denmark, and as Post-Doctoral Researcher and Teaching Assistant at Politecnico di Milano, Italy, where he received Ph.D., MSc. and BSc. in Mechanical Engineering. His research interests include stochastic processes, algorithms for uncertainty quantification, machine learning, and system health management. He* *is member of AIAA, IEEE, and member of the Editorial Board of the PHM Society.*