A Levenberg-Marquardt Algorithm for Sparse Identification of Dynamical Systems

Mark Haring, Esten Ingar Grøtli, *Member, IEEE*, Signe Riemer-Sørensen, Katrine Seel, and Kristian Gaustad Hanssen

***Abstract*— Low complexity of a system model is essential for its use in real-time applications. However, sparse identification methods commonly have stringent requirements that exclude them from being applied in an industrial setting. In this article, we introduce a flexible method for the sparse identification of dynamical systems described by ordinary differential equations. Our method relieves many of the requirements imposed by other methods that relate to the structure of the model and the dataset, such as fixed sampling rates, full state measurements, and linearity of the model. The Levenberg–Marquardt algorithm is used to solve the identification problem. We show that the Levenberg–Marquardt algorithm can be written in a form that enables parallel computing, which greatly diminishes the time required to solve the identification problem. An efficient backward elimination strategy is presented to construct a lean system model.**

***Index Terms*— Artificial neural networks, Levenberg-Marquardt algorithm, machine learning, sparse identification, system identification.**

I. Introduction

IN MANY practical applications, a mathematical model of a dynamical system is a prerequisite to understanding, predicting, and manipulating the behavior of the system in an effective manner. Obtaining a suitable model can be a challenging task. For some systems, first principles (e.g., established laws of physics) may be used to derive model equations that represent the dominant system dynamics. However, for many other systems, this way of model discovery is out of the question due to insufficient information about the system. Moreover, even if first principles can be used to obtain a model, the resulting model may be too complex to use in real-time applications for design, optimization, and control. For example, in areas such as fluid dynamics, electrodynamics, and quantum mechanics, a first-principle model may consist of multiple partial differential equations for which the evolution of the state of the system is hard to compute, let alone in real time.

Alternatively, we may construct a system model using data following a system identification or machine learning approach. The corresponding system model can often be written in the form of an artificial neural network, where the type of neurons can range from commonly used neurons with sigmoidal or rectified linear activation functions to wavelets, splines, radial basis functions, monomials, or Gaussian processes, to name a few [31]. Generally designed to be a universal function approximator, the use of an artificial neural network often leads to an unnecessarily complex model. High complexity strongly hampers the use of the model in real-time applications, and it commonly goes hand in hand with stringent data requirements. It should be noted that the complexity of the model strongly depends on the chosen coordinate frame and type of neurons. As a first example, the orbits of the planets in our solar system can be described much simpler and in fewer terms by taking the Sun as center of reference than Earth. As mentioned in [2] and [29], many physical systems allow for a sparse representation in a suitable coordinate frame. As a second example, it takes a large artificial network with rectified linear units to accurately approximate a quadratic function on a large (but finite) domain, while it requires only three monomials to do the same (i.e., monomials of degrees zero, one, and two, respectively). Training such a large artificial network needs many measurements of the quadratic function, while fitting the three monomials requires a minimum of only three data points (disregarding measurement noise). Without any knowledge of the system to go by, the chance that a chosen coordinate frame and neuron type result in an accurate model of low complexity is generally slim.

In many industrial applications, neither a first-principle model nor a purely data-driven model is constructed with high accuracy due to various reasons including unknown environmental conditions, a varying composition of raw materials, knowledge caveats, imprecise data registration, and a lack of exploration. In addition, there may be parts of the system for which only a few measurements are available due to a high cost or inability to conduct more. What complicates matters further is that these measurements may have irregular sampling rates because smart sampling algorithms (see [22])

Manuscript received January 14, 2021; revised August 14, 2021 and November 29, 2021; accepted February 28, 2022. This work was supported by the industry partners Borregaard, Elkem, Hydro, Yara and by the Research Council of Norway through the project TAPI: Towards Autonomy in Process Industries, under Project 294544. *(Corresponding author: Mark Haring.)*

Mark Haring, Esten Ingar Grøtli, and Kristian Gaustad Hanssen are with the Department of Mathematics and Cybernetics, SINTEF Digital, 7031 Trondheim, Norway (e-mail: mark.haring@sintef.no; esteningar.grotli@sintef.no; kristian.gaustad.hanssen@sintef.no).

Signe Riemer-Sørensen is with the Department of Mathematics and Cybernetics, SINTEF Digital, 0373 Oslo, Norway (email: signe.riemer-sorensen@sintef.no).

Katrine Seel is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), 7491 Trondheim, Norway, and also with the Department of Mathematics and Cybernetics, SINTEF Digital, 7031 Trondheim, Norway (e-mail: katrine.seel@sintef.no).

Digital Object Identifier 10.1109/TNNLS.2022.3157963

have been applied or measurements have been conducted manually. To compose accurate models, one is often forced to combine both first-principle information and measurement data. These resulting gray-box models come in many different forms [15]. In an attempt to improve the model accuracy while keeping the model complexity low, we can use an artificial neural network to model only those parts of the system for which no first-principle model is available or for which the solution of the first-principle model is difficult to compute or inaccurate. However, due to a high network complexity, this alone is generally not enough to ensure that the resulting model is simple enough to be used for real-time applications.

In this article, we apply sparse identification to derive a low-complexity network model. This article is inspired by the works in [2] and [29]. In these, a lean feedforward network is obtained by removing all edges (i.e., the connections between the neurons in the network) that have little or no influence on the quality of the fit. The network can be pruned by adding a sparsity-promoting regularization term to the cost function (or loss function) that is used to train the network; see lasso [33] and sparse relaxed regularized regression [36], for instance. Alternatively, a sequential algorithm, such as stepwise regression [6] or sequential thresholding [2], may be applied to remove all irrelevant edges in an iterative fashion. Other model-selection approaches are discussed in [7] and [20]. The sparse-identification methods in [2] and [29] rely on linear regression. Although they are relatively simple and fast enough to perform model identification in real time [9], [25], they rely on the limiting assumptions that the network model is linear in all its parameters, and that all system states are measured and their first-order time derivatives can be accurately approximated. This last condition commonly translates to the requirement of a sufficiently high sampling rate for all measurements to mitigate the negative influences of measurement noise by averaging over multiple samples. In [19], the computation of time derivatives of the states is based on Koopman theory. This technique can be applied to low-rate sampled data, but the requirement of a fixed sampling rate is often too restrictive to be applied to industrial data.

If the model is composed of differential equations, we may apply numerical integration to compute the state values of the system at any point in time and compare these to the measurements to optimize the model. This removes the requirement that the network should be linear in its parameters. Moreover, an arbitrarily sampled dataset can be used. In [30], the solutions of differential equations modeled by deep neural networks are computed using a Runge–Kutta scheme. In [26], the numerical integration algorithm is directly encoded in the kernels of the Gaussian processes. As an alternative to traditional numerical integration, a neural network can be trained to produce solutions of differential equations [11], [14]. The deep neural network in [27] is designed to obtain solutions of differential equations and identify model parameters. To the best of our knowledge, no attempts of network sparsification have been made for any of the identification methods that rely on numerical integration.

A possible reason for this is the high-computational demand for training the network. Numerical integration of the state equations implies calculating the state values on a finite (and flexible) time grid. Evaluating the cost function used for training the network subsequently requires computing the state values for all grid points. The effective dimension of the optimization problem to be solved for training the network is equal to the number of network parameters plus the number of state variables times the number of grid points. A large number of grid points may be required to accurately approximate the state solution, especially if the time series of measurements span a large time interval. Therefore, the dimension of the optimization problem may be very large. In turn, the optimization problem may be difficult to solve. It is noted that network pruning would require solving an even harder regularized optimization problem or retraining the network multiple times using a backward elimination strategy.

In this article, we develop an efficient method for training an artificial neural network for system models described by ordinary differential equations. Common training algorithms include gradient descent, the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm, and the Levenberg–Marquardt algorithm [12], [21], [32]. Compared to the first-order convergence of gradient descent, a faster second-order convergence can be obtained using L-BFGS or Levenberg–Marquardt. The Levenberg–Marquardt algorithm has the advantage that it is relatively easy to parallelize. Due to the Markovian nature of the system model, we can rewrite the approximation of the optimization problem by Levenberg and Marquardt as batches of recursive small-scale optimization subproblems. We effectively divide the time interval from the first to the last measurement in separate subintervals, where each subinterval corresponds to one batch of optimization subproblems. All batches can be solved in parallel. Subsequently, the solution to the original approximation can be reconstructed by combining the results from all batches. Parallelization can significantly reduce the wall time required to train the network. In addition, we outline how a similar problem formulation can be used to prune the network using backward elimination by removing irrelevant network edges one at a time.

This article is organized as follows. A formulation of the system identification problem is given in Section II. The standard approach of solving the system identification problem using the Levenberg–Marquardt algorithm is discussed in Section III. In Section IV, it is shown how the solution method can be rewritten to parallelize parts of the algorithm. In Section V, a method for network sparsification using backward elimination is presented that utilizes a similar approach as outlined in Section IV to efficiently approximate the effect of removing a network edge on the overall quality of the fit. In Section VI, the sparsification method is illustrated by means of two examples. Advantages and drawbacks of the proposed methodology are summarized in Section VII.

A. Notations

The sets of real numbers, nonnegative real numbers, and positive real numbers are denoted by $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, and $\mathbb{R}_{>0}$, respectively. $\mathbb{N}$ and $\mathbb{N}_{>0}$ denote the sets of natural numbers

(i.e. nonnegative integers) and positive integers. The ceiling function that maps a real number α to the least integer greater than or equal to α is denoted by $\lceil \alpha \rceil$. The transpose of any vector $\mathbf{x}$ is written as $\mathbf{x}^T$. For any nonsingular, square matrix $\mathbf{A}$, its inverse is denoted by $\mathbf{A}^{-1}$. Similarly, $\mathbf{A}^{-T}$ is the transpose of the inverse. The Moore–Penrose pseudoinverse of the matrix $\mathbf{M}$ is written as $\mathbf{M}^+$. We denote the union of two sets $\mathcal{B}$ and $\mathcal{C}$ by $\mathcal{B} \cup \mathcal{C}$. If $\mathcal{C}$ is a subset of $\mathcal{B}$, then the set difference of $\mathcal{B}$ and $\mathcal{C}$ (i.e. the relative complement of $\mathcal{C}$ in $\mathcal{B}$) is written as $\mathcal{B} \setminus \mathcal{C}$. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be an ordered set with n elements. The notation $\{\mathbf{y}(t)\}_{t \in \mathcal{T}}$ is short for $\{\mathbf{y}(t_1), \mathbf{y}(t_2), \ldots, \mathbf{y}(t_n)\}$. For any time-varying vector signal $\mathbf{s}(t)$, where t denotes the time, we write $(d\mathbf{s}/dt)(t)$ as $\dot{\mathbf{s}}(t)$. The Euclidean norm is denoted by $\|\cdot\|$.

II. Formulation of the System Identification Problem

Let the state trajectories of a dynamical system be given by the solutions of the following ordinary-differential equations that comprise a first-principle model:

$$\dot{\mathbf{x}}(t) = \mathbf{f}_{\text{phys}}(t, \mathbf{x}(t)) + \mathbf{e}(t). \tag{1}$$

Here, $\mathbf{x}(t) \in \mathbb{R}^{n_\mathbf{x}}$ is the state vector with dimension $n_\mathbf{x} \in \mathbb{N}_{>0}$, $\mathbf{f}_{\text{phys}} : \mathbb{R} \times \mathbb{R}^{n_\mathbf{x}} \to \mathbb{R}^{n_\mathbf{x}}$ is a continuously differentiable function, $\mathbf{e}(t) \in \mathbb{R}^{n_\mathbf{x}}$ is the model error, and $t \in \mathbb{R}$ denotes the time. The exact value of the state is unknown in most practical scenarios. To improve the accuracy of the first-principle model, we approximate the error by a continuously differentiable feedforward network $\mathbf{f}_{\text{net}} : \mathbb{R}^{n_\mathbf{x}} \times \mathbb{R}^{n_\mathbf{u}} \times \mathbb{R}^{n_\mathbf{a}} \to \mathbb{R}^{n_\mathbf{x}}$. That is,

$$\mathbf{e}(t) \approx \mathbf{f}_{\text{net}}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{a}). \tag{2}$$

The network consists of an input layer, one or multiple hidden layers, and an output layer. The inputs of the network are the state $\mathbf{x}(t)$ and the vector of known time-varying functions $\mathbf{u}(t) \in \mathbb{R}^{n_\mathbf{u}}$ with dimension $n_\mathbf{u} \in \mathbb{N}$. The output of the network is the estimate of the error in the first-principle model. The vector of network parameters (e.g., weights and biases) is given by $\mathbf{a} \in \mathbb{R}^{n_\mathbf{a}}$, where $n_\mathbf{a} \in \mathbb{N}_{>0}$ is its dimension. We do not pose any restrictions on the structure of the network or the choice of neurons. It is noted that this formulation can be easily adapted to include other kinds of error models, such as the closure models in [23]. Although the focus of this article is to improve the accuracy of the first-principle model by approximating the model error with a feedforward network, we note that this formulation can be made more general by considering models of the form

$$\dot{\mathbf{x}}(t) \approx \mathbf{f}(t, \mathbf{x}(t), \mathbf{a}) \tag{3}$$

where $\mathbf{a}$ is allowed to be any parameter vector. The only essential requirement is that the function $\mathbf{f} : \mathbb{R} \times \mathbb{R}^{n_\mathbf{x}} \times \mathbb{R}^{n_\mathbf{a}} \to \mathbb{R}^{n_\mathbf{x}}$ is differentiable with respect to the state and parameter vectors. Obviously, the system model consisting of the first-principle model in (1) and the error model in (2) can be written in this form.

To identify suitable parameter values, we compare the state solutions of the model with process measurements. Consider a finite time series of process measurements. Let $\mathcal{T}_m \subset \mathbb{R}$ be a finite set containing all measurement times. We denote by $\mathbf{y}(t_m) \in \mathbb{R}^{n_\mathbf{y}}$ the vector of measurements with dimension $n_\mathbf{y} \in \mathbb{N}_{>0}$ taken at time $t = t_m \in \mathcal{T}_m$. The dimension $n_\mathbf{y}$ is allowed to be time varying [i.e., $n_\mathbf{y} = n_\mathbf{y}(t_m)$]. The relation between the measurements and the state is modeled by a continuously differentiable function $\mathbf{h} : \mathcal{T}_m \times \mathbb{R}^{n_\mathbf{x}} \to \mathbb{R}^{n_\mathbf{y}}$, such that

$$\mathbf{y}(t_m) \approx \mathbf{h}(t_m, \mathbf{x}(t_m)) \tag{4}$$

for all $t = t_m \in \mathcal{T}_m$. It is noted that the sampling of the measurements can be arbitrary. Notation-wise, it is convenient to number the measurements. We define

$$\mathcal{I}_m = \{i \in \mathbb{N} : 1 \leq i \leq N_m\} \tag{5}$$

where $N_m \in \mathbb{N}_{>0}$ is the cardinality of $\mathcal{T}_m$ (i.e., the set $\mathcal{T}_m$ contains N_m elements). For any $i \in \mathcal{I}_m$, $t_m^{(i)}$ denotes the i^{th} element of the set $\mathcal{T}_m$ in ascending order. In turn, the i^{th} vector of measurements is denoted by $\mathbf{y}(t_m^{(i)})$.

Obtaining the parameter values that fit the measurements best requires that the state and the parameters of the model are identified simultaneously. To find suitable values, we compute the minimizer of the nonlinear least-squares problem

$$\begin{aligned}\min_{\mathbf{a},\, \{\mathbf{x}\},\, \{\boldsymbol{\varepsilon}_\mathbf{x}\},\, \{\boldsymbol{\varepsilon}_\mathbf{y}\}} \Bigg\{ & \int_{\underline{t}_m}^{\overline{t}_m} \boldsymbol{\varepsilon}_\mathbf{x}^T(t) \mathbf{W}_\mathbf{x}^{-1}(t) \boldsymbol{\varepsilon}_\mathbf{x}(t) dt + \mu_\mathbf{a} \|\mathbf{a}\|^2 \\ & + \sum_{t_m \in \mathcal{T}_m} \boldsymbol{\varepsilon}_\mathbf{y}^T(t_m) \mathbf{W}_\mathbf{y}^{-1}(t_m) \boldsymbol{\varepsilon}_\mathbf{y}(t_m) \\ & + \mu_\mathbf{x} \int_{\underline{t}_m}^{\overline{t}_m} \|\mathbf{x}(t)\|^2 dt : \\ & \big(\forall t \in \big[\underline{t}_m, \overline{t}_m\big]\big)[\dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}(t), \mathbf{a}) + \boldsymbol{\varepsilon}_\mathbf{x}(t)] \\ & (\forall t_m \in \mathcal{T}_m)\big[\mathbf{y}(t_m) = \mathbf{h}(t_m, \mathbf{x}(t_m)) + \boldsymbol{\varepsilon}_\mathbf{y}(t_m)\big] \Bigg\}\end{aligned} \tag{6}$$

where $\{\mathbf{x}\}$ and $\{\boldsymbol{\varepsilon}_\mathbf{x}\}$ are shorthand notations for the state $\{\mathbf{x}(t)\}_{t=\underline{t}_m}^{\overline{t}_m}$ and the state error $\{\boldsymbol{\varepsilon}_\mathbf{x}(t)\}_{t=\underline{t}_m}^{\overline{t}_m}$, respectively, and where $\{\boldsymbol{\varepsilon}_\mathbf{y}\}$ is short for the set $\{\boldsymbol{\varepsilon}_\mathbf{y}(t_m)\}_{t_m \in \mathcal{T}_m}$, with modeling error $\boldsymbol{\varepsilon}_\mathbf{y}(t_m) \in \mathbb{R}^{n_\mathbf{y}}$ for all $t_m \in \mathcal{T}_m$. Here, $\underline{t}_m = t_m^{(1)}$ and $\overline{t}_m = t_m^{(N_m)}$ are the times of the first and last measurement, respectively. The symmetric, positive-definite matrices $\mathbf{W}_\mathbf{x}(t) \in \mathbb{R}^{n_\mathbf{x} \times n_\mathbf{x}}$ and $\mathbf{W}_\mathbf{y}(t_m) \in \mathbb{R}^{n_\mathbf{y} \times n_\mathbf{y}}$ are weighting matrices of the model errors $\boldsymbol{\varepsilon}_\mathbf{x}(t)$ and $\boldsymbol{\varepsilon}_\mathbf{y}(t_m)$, respectively. If $\boldsymbol{\varepsilon}_\mathbf{x}(t)$ and $\boldsymbol{\varepsilon}_\mathbf{y}(t_m)$ are regarded as vectors of stochastic variables, then $\mathbf{W}_\mathbf{x}(t)$ and $\mathbf{W}_\mathbf{y}(t_m)$ can be chosen equal to the covariance matrix of $\boldsymbol{\varepsilon}_\mathbf{x}(t)$ and $\boldsymbol{\varepsilon}_\mathbf{y}(t_m)$ to obtain an estimate with minimal variance. A more pragmatic approach is to think of $\mathbf{W}_\mathbf{x}(t)$ and $\mathbf{W}_\mathbf{y}(t_m)$ as scaling matrices, such that $\mathbf{S}_\mathbf{x}^{-1}(t)\boldsymbol{\varepsilon}_\mathbf{x}(t)$ and $\mathbf{S}_\mathbf{y}^{-T}(t_m)\boldsymbol{\varepsilon}_\mathbf{y}(t_m)$ are vectors of elements with roughly equal magnitude based on prior knowledge, where

$$\mathbf{W}_\mathbf{x}(t) = \mathbf{S}_\mathbf{x}(t)\mathbf{S}_\mathbf{x}^T(t) \tag{7}$$

and

$$\mathbf{W}_\mathbf{y}(t_m) = \mathbf{S}_\mathbf{y}(t_m)\mathbf{S}_\mathbf{y}^T(t_m). \tag{8}$$

The terms in the cost function related to the (small) constants $\mu_\mathbf{x}, \mu_\mathbf{a} \in \mathbb{R}_{\geq 0}$ are regularization terms to remedy the effects

of overfitting [35]. If multiple or even infinitely many minima exist, these regularization terms keep the magnitudes of the computed, minimizing state and parameter values relatively small. The selection of suitable regularization constants is an open problem, and a topic for future work.

III. Solving the System Identification Problem Using the Levenberg–Marquardt Algorithm

To compute the solution of the optimization problem in (6), we first discretize the problem in time. Let $\Delta t \in \mathbb{R}_{>0}$ be the maximal step size of the discretization. It is noted that all continuous-time signals in (6) are defined on the interval $[\underline{t}_m, \overline{t}_m]$. Let $\mathcal{T}_d \subset \mathbb{R}$ be a set of $N_d \in \mathbb{N}_{>0}$ discretization points that are chosen such that $\mathcal{T}_m \subseteq \mathcal{T}_d$, and the distance between all subsequent points is smaller than or equal to Δt. A routine to generate such set is outlined in Appendix A. To make the proceeding notations easier, we define the index set

$$\mathcal{J}_d = \{j \in \mathbb{N} : 1 \leq j \leq N_d\}. \tag{9}$$

For any $j \in \mathcal{J}_d$, we denote by $t_d^{(j)}$ the j^{th} element of $\mathcal{T}_d$ in ascending order. In addition, we define the index set

$$\mathcal{J}_m = \left\{ j \in \mathcal{J}_d : (\exists t_m \in \mathcal{T}_m)\left[t_d^{(j)} = t_m\right]\right\} \tag{10}$$

such that $t_d^{(j_m)} \in \mathcal{T}_m$ for all $j_m \in \mathcal{J}_m$. The (variable) discretization step size is given by

$$\Delta t_d^{\left(j+\frac{1}{2}\right)} = t_d^{(j+1)} - t_d^{(j)} \tag{11}$$

for all $j \in \mathcal{J}_d \setminus \{N_d\}$.

We discretize the least-squares problem in (6) using the midpoint rule and the trapezoidal rule. With a minor abuse of notation, we get the optimization problem in (12), as shown at the bottom of the page, with

$$t_d^{\left(j+\frac{1}{2}\right)} = \frac{1}{2}\left(t_d^{(j+1)} + t_d^{(j)}\right) \tag{13}$$

for all $j \in \mathcal{J}_d \setminus \{N_d\}$. The sets $\{\mathbf{x}\}$, $\{\boldsymbol{\varepsilon}_\mathbf{x}\}$ and $\{\boldsymbol{\varepsilon}_\mathbf{y}\}$ are now short for $\{\mathbf{x}(t_d^{(j)})\}_{j\in\mathcal{J}_d}$, $\{\boldsymbol{\varepsilon}_\mathbf{x}(t_d^{(j)})\}_{j\in\mathcal{J}_d}$ and $\{\boldsymbol{\varepsilon}_\mathbf{y}(t_d^{(j_m)})\}_{j_m\in\mathcal{J}_m}$, respectively. Because the discretization errors of the midpoint rule and the trapezoidal rule for each discretization step are $\mathcal{O}(\Delta t^3)$ and the total number of discretization steps is $\mathcal{O}(\Delta t^{-1})$, the corresponding difference between the solutions of (6) and (12) for all discretization steps combined is $\mathcal{O}(\Delta t^2)$. Hence, the solutions are identical in the limit as Δt approaches zero.

Remark 1: The midpoint rule and the trapezoidal rule are not the only numerical integration methods that can be applied to discretize the optimization problem in (6). However, any such integration method is required to be explicit to apply the Levenberg–Marquardt algorithm later in this section. Alternatively, we may change the problem formulation in (6) by replacing all integrals by finite, weighted sums of function evaluations, as in [30], for instance. However, the solution of this altered optimization problem may not correspond to the solution of (6), not even in the limit.

We may simplify the optimization problem in (12) by eliminating the variables of the model errors $\{\boldsymbol{\varepsilon}_\mathbf{x}\}$ and $\{\boldsymbol{\varepsilon}_\mathbf{y}\}$ using the constraint equations. Subsequently, the optimization problem can be written as

$$\min_{\mathbf{b}} \|\mathbf{g}(\mathbf{b})\|^2 \tag{14}$$

with

$$\mathbf{b} = \left[\mathbf{x}^T\left(t_d^{(1)}\right),\ \mathbf{x}^T\left(t_d^{(2)}\right),\ \ldots,\ \mathbf{x}^T\left(t_d^{(N_d)}\right),\ \mathbf{a}^T\right]^T \tag{15}$$

and

$$\mathbf{g}(\mathbf{b}) = \begin{bmatrix} \mathbf{p}_{\mathbf{x},1}(\mathbf{x}(t_d^{(1)}), \mathbf{x}(t_d^{(2)}), \mathbf{a}) \\ \mathbf{p}_{\mathbf{x},2}(\mathbf{x}(t_d^{(2)}), \mathbf{x}(t_d^{(3)}), \mathbf{a}) \\ \vdots \\ \mathbf{p}_{\mathbf{x},N_d-1}(\mathbf{x}(t_d^{(N_d-1)}), \mathbf{x}(t_d^{(N_d)}), \mathbf{a}) \\ \mathbf{p}_{\mathbf{x},N_d}(\mathbf{x}(t_d^{(N_d)})) \\ \mathbf{p}_\mathbf{a}(\mathbf{a}) \end{bmatrix} \tag{16}$$

where as (17), shown at the bottom of the next page, for all $j \in \mathcal{J}_d \setminus \{N_d\}$

$$\mathbf{p}_{\mathbf{x},N_d}(\mathbf{x}) = \mathbf{S}_\mathbf{y}^{-1}\left(t_d^{(N_d)}\right)\left(\mathbf{y}\left(t_d^{(N_d)}\right) - \mathbf{h}\left(t_d^{(N_d)}, \mathbf{x}\right)\right) \tag{18}$$

and

$$\mathbf{p}_\mathbf{a}(\mathbf{a}) = \sqrt{\mu_\mathbf{a}}\mathbf{a}. \tag{19}$$

The optimization problem in (14) may be solved iteratively. Let $k \in \mathbb{N}$ be the iteration number. Writing the values of the optimization variables in (15) at iteration k as $\mathbf{b}_k$, the first-order Taylor series approximation

$$\mathbf{g}(\mathbf{b}_{k+1}) \approx \mathbf{g}(\mathbf{b}_k) + \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\boldsymbol{\gamma}_k \tag{20}$$

is accurate if $\boldsymbol{\gamma}_k$ is sufficiently small, where $\boldsymbol{\gamma}_k$ is defined as

$$\boldsymbol{\gamma}_k = \mathbf{b}_{k+1} - \mathbf{b}_k. \tag{21}$$

$$\min_{\mathbf{a},\{\mathbf{x}\},\{\boldsymbol{\varepsilon}_\mathbf{x}\},\{\boldsymbol{\varepsilon}_\mathbf{y}\}} \left\{ \sum_{j\in\mathcal{J}_d\setminus\{N_d\}} \boldsymbol{\varepsilon}_\mathbf{x}^T\left(t_d^{\left(j+\frac{1}{2}\right)}\right)\mathbf{W}_\mathbf{x}^{-1}\left(t_d^{\left(j+\frac{1}{2}\right)}\right)\boldsymbol{\varepsilon}_\mathbf{x}\left(t_d^{\left(j+\frac{1}{2}\right)}\right)\Delta t_d^{\left(j+\frac{1}{2}\right)} + \sum_{j_m\in\mathcal{J}_m} \boldsymbol{\varepsilon}_\mathbf{y}^T\left(t_d^{(j_m)}\right)\mathbf{W}_\mathbf{y}^{-1}\left(t_d^{(j_m)}\right)\boldsymbol{\varepsilon}_\mathbf{y}\left(t_d^{(j_m)}\right) + \mu_\mathbf{a}\|\mathbf{a}\|^2 \right.$$
$$+\frac{\mu_\mathbf{x}}{2}\sum_{j\in\mathcal{J}_d\setminus\{N_d\}}\left(\left\|\mathbf{x}\left(t_d^{(j)}\right)\right\|^2 + \left\|\mathbf{x}\left(t_d^{(j+1)}\right)\right\|^2\right)\Delta t_d^{\left(j+\frac{1}{2}\right)} : (\forall j_m \in \mathcal{J}_m)\left[\mathbf{y}\left(t_d^{(j_m)}\right) = \mathbf{h}\left(t_d^{(j_m)}, \mathbf{x}\left(t_d^{(j_m)}\right)\right) + \boldsymbol{\varepsilon}_\mathbf{y}\left(t_d^{(j_m)}\right)\right]$$
$$\left.(\forall j \in \mathcal{J}_d \setminus \{N_d\})\left[\frac{\mathbf{x}\left(t_d^{(j+1)}\right) - \mathbf{x}\left(t_d^{(j)}\right)}{\Delta t_d^{\left(j+\frac{1}{2}\right)}} = \mathbf{f}\left(t_d^{\left(j+\frac{1}{2}\right)}, \tfrac{1}{2}\left(\mathbf{x}\left(t_d^{(j)}\right) + \mathbf{x}\left(t_d^{(j+1)}\right)\right), \mathbf{a}\right) + \boldsymbol{\varepsilon}_\mathbf{x}\left(t_d^{\left(j+\frac{1}{2}\right)}\right)\right]\right\} \tag{12}$$

Applying the approximation in (20) and the change of variables in (21), we obtain the Gauss–Newton method

$$\min_{\mathbf{b}_{k+1}} \|\mathbf{g}(\mathbf{b}_{k+1})\|^2 \approx \min_{\boldsymbol{\gamma}_k} \left\| \mathbf{g}(\mathbf{b}_k) + \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\boldsymbol{\gamma}_k \right\|^2. \tag{22}$$

To prevent $\boldsymbol{\gamma}_k$ from being too large, Levenberg [13] and Marquardt [18] propose to add a regularization term (also denoted as damping term) to the cost function, resulting in

$$\min_{\boldsymbol{\gamma}_k} \left\{ \left\| \mathbf{g}(\mathbf{b}_k) + \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\boldsymbol{\gamma}_k \right\|^2 + \lambda_k \left\| \boldsymbol{\gamma}_k \right\|^2 \right\} \tag{23}$$

where $\lambda_k \in \mathbb{R}_{>0}$ is a tuning parameter. Noting that (23) is a linear least-squares problem, the minimizer of the optimization problem is given by

$$\boldsymbol{\gamma}_k = -\left(\left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) \right)^T \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) + \lambda_k \mathbf{I} \right)^{-1} \times \left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) \right)^T \mathbf{g}(\mathbf{b}_k) \tag{24}$$

see, for example, [4, Th. 2.1.1]. By combining (21) and (24), we get the update law

$$\mathbf{b}_{k+1} = \mathbf{b}_k - \left(\left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) \right)^T \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) + \lambda_k \mathbf{I} \right)^{-1} \times \left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) \right)^T \mathbf{g}(\mathbf{b}_k). \tag{25}$$

Hence, the new values $\mathbf{b}_{k+1}$ are equal to the old values $\mathbf{b}_k$ minus the product of a symmetric, positive-definite matrix and the gradient of the cost function in (14). It is noted that the update law in (25) is similar to that of the Gauss–Newton method for small values of λ_k. Moreover, the search direction in which the optimization variables are updated is similar to the gradient-descent direction for large values of λ_k. Due to the uncertainty in the linearized model far from the linearization point, convergence of the algorithm cannot be guaranteed for small values of λ_k. On the other hand, the decrease in the value of the cost function is small for large values of λ_k. Therefore, it may take many steps to converge. Various algorithms have been proposed to balance the uncertainty and magnitude of the decrease in cost function value; see [16], [24], [34] and references therein. Algorithm 1 is a simple illustration of an adaptation method for λ_k. While the gradient of the cost function is larger than a small constant $\sigma > 0$, new values of the optimization variables are generated in Line 3 in accordance with (25). If these new values lead to a decrease in cost-function value, the values are accepted and λ_k is decreased by a factor $(1/\rho_1)$. If not, the values are rejected and λ_k is increased by a factor ρ_2, where $\rho_2 \geq \rho_1 > 1$. It is noted that robustness measures, such as a positive lower bound on the decrease of the cost-function value in Line 4 and an upper bound on the maximal number of iterations, should be added for practical applications of the algorithm [16]. Moreover, the optimization method by Levenberg and Marquardt only converges to local optima.

Algorithm 1 Standard Levenberg–Marquardt Algorithm

Input: $\mathbf{b}_0$, λ_0, (parameters: σ, ρ_1, ρ_2)
Output: $\mathbf{b}_k$

1: $k \leftarrow 0$
2: **while** $\|2\left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\right)^T \mathbf{g}(\mathbf{b}_k)\| > \sigma$ **do**
3: $\quad \mathbf{b}'_{k+1} \leftarrow \mathbf{b}_k - \left(\left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\right)^T \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k) + \lambda_k \mathbf{I} \right)^{-1} \times \left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\right)^T \mathbf{g}(\mathbf{b}_k)$
4: $\quad$ **if** $\|\mathbf{g}(\mathbf{b}'_{k+1})\|^2 < \|\mathbf{g}(\mathbf{b}_k)\|^2$ **then**
5: $\quad\quad \mathbf{b}_{k+1} \leftarrow \mathbf{b}'_{k+1}$
6: $\quad\quad \lambda_{k+1} \leftarrow \frac{\lambda_k}{\rho_1}$
7: $\quad$ **else**
8: $\quad\quad \mathbf{b}_{k+1} \leftarrow \mathbf{b}_k$
9: $\quad\quad \lambda_{k+1} \leftarrow \rho_2 \lambda_k$
10: $\quad$ **end if**
11: $\quad k \leftarrow k + 1$
12: **end while**

IV. Exploiting the Structure of the Optimization Problem to Enable Parallelization

If the measurement times span a large interval (i.e., $\overline{t}_m - \underline{t}_m$ is large) and the maximal discretization step Δt is small, the number of discretization points N_d is very large. Therefore, the number of variables of the optimization problem in (23), which is equal to the dimension $n_{\mathbf{b}} = n_{\mathbf{a}} + n_{\mathbf{x}} \times N_d$ of the vector $\mathbf{b}$ in (15), can be very large. In turn, inverting a very large matrix to compute the update in (25) can be costly. In this section, we present a method to parallelize the computation of the solution of the optimization problem in (23) to speed up computations.

$$\mathbf{p}_{\mathbf{x},j}(\mathbf{x},\mathbf{z},\mathbf{a}) = \begin{bmatrix} \mathbf{S}_{\mathbf{x}}^{-1}\left(t_d^{(j+\frac{1}{2})}\right)\left(\dfrac{\mathbf{z}-\mathbf{x}}{\Delta t_d^{(j+\frac{1}{2})}} - \mathbf{f}\left(t_d^{(j+\frac{1}{2})}, \tfrac{1}{2}(\mathbf{x}+\mathbf{z}), \mathbf{a}\right)\right)\sqrt{\Delta t_d^{(j+\frac{1}{2})}} \\ \sqrt{\dfrac{\mu_{\mathbf{x}}}{2}}\mathbf{x}\sqrt{\Delta t_d^{(j+\frac{1}{2})}} \\ \sqrt{\dfrac{\mu_{\mathbf{x}}}{2}}\mathbf{z}\sqrt{\Delta t_d^{(j+\frac{1}{2})}} \\ \begin{cases} \mathbf{S}_{\mathbf{y}}^{-1}\left(t_d^{(j)}\right)\left(\mathbf{y}\left(t_d^{(j)}\right) - \mathbf{h}\left(t_d^{(j)}, \mathbf{x}\right)\right), & \text{if } j \in \mathcal{J}_m \\ 0, & \text{if } j \notin \mathcal{J}_m \end{cases} \end{bmatrix} \tag{17}$$

Let us define

$$\boldsymbol{\beta}_k\left(t_d^{(j)}\right) = \mathbf{x}_{k+1}\left(t_d^{(j)}\right) - \mathbf{x}_k\left(t_d^{(j)}\right), \quad \boldsymbol{\delta}_k = \mathbf{a}_{k+1} - \mathbf{a}_k \tag{26}$$

for all $j \in \mathcal{J}_m$, such that $\boldsymbol{\gamma}_k$ in (21) is given by

$$\boldsymbol{\gamma}_k = \left[\boldsymbol{\beta}_k^T\left(t_d^{(1)}\right),\ \boldsymbol{\beta}_k^T\left(t_d^{(2)}\right),\ \ldots,\ \boldsymbol{\beta}_k^T\left(t_d^{(N_d)}\right),\ \boldsymbol{\delta}_k^T\right]^T. \tag{27}$$

It follows that the optimization problem in (23) can be written as

$$\min_{\{\boldsymbol{\beta}_k\},\boldsymbol{\delta}_k}\left\{\sum_{j=1}^{N_d} q_{j,k} + r_k\right\} \tag{28}$$

with

$$q_{j,k} = \left\|\mathbf{p}_{\mathbf{x},j} + \frac{\partial \mathbf{p}_{\mathbf{x},j}}{\partial \mathbf{x}}\boldsymbol{\beta}_k\left(t_d^{(j)}\right) + \frac{\partial \mathbf{p}_{\mathbf{x},j}}{\partial \mathbf{z}}\boldsymbol{\beta}_k\left(t_d^{(j+1)}\right) + \frac{\partial \mathbf{p}_{\mathbf{x},j}}{\partial \mathbf{a}}\boldsymbol{\delta}_k\right\|^2 + \lambda_k\left\|\boldsymbol{\beta}_k\left(t_d^{(j)}\right)\right\|^2 \tag{29}$$

for all $j \in \mathcal{J}_d \setminus \{N_d\}$,

$$q_{N_d,k} = \left\|\mathbf{p}_{\mathbf{x},N_d} + \frac{\partial \mathbf{p}_{\mathbf{x},N_d}}{\partial \mathbf{x}}\boldsymbol{\beta}_k\left(t_d^{(N_d)}\right)\right\|^2 + \lambda_k\left\|\boldsymbol{\beta}_k\left(t_d^{(N_d)}\right)\right\|^2 \tag{30}$$

and

$$r_k = \left\|\mathbf{p}_{\mathbf{a}} + \frac{d\mathbf{p}_{\mathbf{a}}}{d\mathbf{a}}\boldsymbol{\delta}_k\right\|^2 + \lambda_k\|\boldsymbol{\delta}_k\|^2 \tag{31}$$

where $\{\boldsymbol{\beta}_k\}$ is short for $\{\boldsymbol{\beta}_k(t_d^{(j)})\}_{j\in\mathcal{J}_d}$. We have omitted the arguments of the functions $\mathbf{p}_{\mathbf{x},j}$ and $\mathbf{p}_{\mathbf{a}}$ and their derivatives to shorten the expressions in (28)–(31). It is noted that the corresponding arguments of $\mathbf{p}_{\mathbf{x},j}$ and its derivatives are $(\mathbf{x}_k(t_d^{(j)}), \mathbf{x}_k(t_d^{(j+1)}), \mathbf{a}_k)$ for $j \in \mathcal{J}_d \setminus \{N_d\}$ and $\mathbf{x}_k(t_d^{(N_d)})$ for $j = N_d$, and that the argument of $\mathbf{p}_{\mathbf{a}}$ and its derivative is $\mathbf{a}_k$.

Now, to compute the solution of the optimization problem in (23) using $N_s \in \mathbb{N}_{>0}$ parallel processes, let us divide the sum in (28) in N_s smaller sums

$$\sum_{j=1}^{N_d} q_{j,k} = \sum_{s=1}^{N_s} v_{s,k}, \quad \text{with} \quad v_{s,k} = \sum_{j=\zeta(s)}^{\zeta(s+1)-1} q_{j,k} \tag{32}$$

for all $s \in \{1, 2, \ldots, N_s\}$. Here, $q_{0,k} = 0$, and $\zeta : \mathbb{N} \to \mathbb{N}$ is a strictly increasing function that satisfies $\zeta(1) = 0$ and $\zeta(N_s + 1) = N_d + 1$. The function ζ is chosen such that the number of summands of each partial sum in (32) is roughly the same. Instead of minimizing the cost function in (28) over all optimization variables at once, we minimize the partial sums in (32) over all optimization variables that do not appear in any of the other partial sums (i.e., $\{\boldsymbol{\beta}_k(t_d^{(j)})\}_{j=\zeta(s)+1}^{\zeta(s+1)-1}$ for all $s \in \{1, 2, \ldots, N_s\}$). For each $s \in \{1, 2, \ldots, N_s\}$, this leads to the series of nested optimization subproblems with solution $w_{s,k}$ in (33), as shown at the bottom of the page. Note that $w_{s,k}$ can be rewritten as

$$w_{s,k} = H_{s,\zeta(s+1)-\zeta(s),k} \tag{34}$$

using the recursion

$$H_{s,j+1,k} = \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(s)+j)}\right)}\left\{q_{\zeta(s)+j,k} + H_{s,j,k}\right\} \tag{35}$$

for all $j \in \{1, 2, \ldots, \zeta(s+1) - \zeta(s) - 1\}$, with $H_{s,1,k} = q_{\zeta(s),k}$. Subsequently, the optimization problem in (28) itself can be formulated as the series of nested optimization subproblems; see (36), as shown at the bottom of the page. Alternatively, we can write (36) as

$$\min_{\boldsymbol{\delta}_k}\left\{r_k + G_{N_s,k}\right\} \tag{37}$$

where $G_{N_s,k}$ is obtained by the recursion

$$G_{s,k} = \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(s))}\right)}\left\{w_{s,k} + G_{s-1,k}\right\} \tag{38}$$

for all $s \in \{2, 3, \ldots, N_s\}$, with $G_{1,k} = w_{1,k}$. We are able to write the optimization problem in (23) as nested optimization subproblems due to the Markovian structure of the model in (3)–(4). The approach of solving an optimization problem by recursively solving optimization subproblems is known as dynamic programming [1]. There are two important things to note here. First, all optimization subproblems in (35), (37), and (38) are small-scale problems, depending only on a few optimization variables. Therefore, the operative memory required

$$\begin{aligned} w_{s,k} &= \min_{\left\{\boldsymbol{\beta}_k\left(t_d^{(j)}\right)\right\}_{j=\zeta(s)+1}^{\zeta(s+1)-1}} v_{s,k} = \min_{\left\{\boldsymbol{\beta}_k\left(t_d^{(j)}\right)\right\}_{j=\zeta(s)+1}^{\zeta(s+1)-1}}\left\{\sum_{j=\zeta(s)}^{\zeta(s+1)-1} q_{j,k}\right\} \\ &= \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(s+1)-1)}\right)}\left\{q_{\zeta(s+1)-1,k} + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(s+1)-2)}\right)}\left\{\cdots + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(s)+2)}\right)}\left\{q_{\zeta(s)+2,k} + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(s)+1)}\right)}\left\{q_{\zeta(s)+1,k} + q_{\zeta(s),k}\right\}\right\}\cdots\right\}\right\} \end{aligned} \tag{33}$$

$$\begin{aligned} \min_{\{\boldsymbol{\beta}_k\},\boldsymbol{\delta}_k}\left\{\sum_{j=1}^{N_d} q_{j,k} + r_k\right\} &= \min_{\left\{\beta_k\left(t_d^{(\zeta(s))}\right)\right\}_{s=2}^{N_s},\boldsymbol{\delta}_k}\left\{\sum_{s=1}^{N_s} w_{s,k} + r_k\right\} \\ &= \min_{\boldsymbol{\delta}_k}\left\{r_k + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(N_s))}\right)}\left\{w_{N_s,k} + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(N_s-1))}\right)}\left\{\cdots + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(3))}\right)}\left\{w_{3,k} + \min_{\boldsymbol{\beta}_k\left(t_d^{(\zeta(2))}\right)}\left\{w_{2,k} + w_{1,k}\right\}\right\}\cdots\right\}\right\}\right\} \end{aligned} \tag{36}$$

to compute the solution is much lower than for large-scale problem in (23). Second, for each $s \in \{1, 2, \ldots, N_s\}$, the solution $w_{s,k}$ in (34) can be computed independently and, thus, in parallel. Given sufficient computational power, parallelization greatly reduces the time required to solve the optimization problem. This makes it viable to use much longer measurement series for the identification of the system, which may considerably improve the accuracy of the identified model.

It should be noted that $q_{j,k}$, r_k, $w_{s,k}$, $H_{s,j,k}$, and $G_{s,k}$ are all quadratic, sum-of-squares functions of optimization variables. It follows that the minimizers of the optimization subproblems in (35), (37) and (38) can be computed analytically as linear functions of other optimization variables (i.e., the ones we have not minimized over yet); see Appendix B. To be precise, for any $s \in \{1, 2, \ldots, N_s\}$ and any integer j that satisfies $\zeta(s) < j < \zeta(s+1)$, the minimizer of the subproblem in (35) can be written as

$$\boldsymbol{\beta}_k\left(t_d^{(j)}\right) = \begin{cases} l_{j,k}\left(\boldsymbol{\beta}_k\left(t_d^{(j+1)}\right), \boldsymbol{\delta}_k\right), & \text{if } s = 1 \\ l_{j,k}\left(\boldsymbol{\beta}_k\left(t_d^{(\zeta(N_s))}\right), \boldsymbol{\delta}_k\right), & \text{if } j = N_d \\ l_{j,k}\left(\boldsymbol{\beta}_k\left(t_d^{(\zeta(s))}\right), \boldsymbol{\beta}_k\left(t_d^{(j+1)}\right), \boldsymbol{\delta}_k\right), & \text{otherwise} \end{cases} \tag{39}$$

for some linear function $l_{j,k}$. Similarly, for any $s \in \{2, 3, \ldots, N_s\}$, the minimizer of the subproblem in (38) is given by

$$\boldsymbol{\beta}_k\left(t_d^{(\zeta(s))}\right) = \begin{cases} l_{\zeta(s),k}(\boldsymbol{\delta}_k), & \text{if } s = N_s \\ l_{\zeta(s),k}\left(\boldsymbol{\beta}_k\left(t_d^{(\zeta(s+1))}\right), \boldsymbol{\delta}_k\right), & \text{otherwise} \end{cases} \tag{40}$$

for some linear function $l_{\zeta(s),k}$. Because we have minimized the optimization problem over all other variables, the minimizing values of $\boldsymbol{\delta}_k$ can be directly obtained from (37). We can recursively reconstruct the optimal values of the minimizers $\boldsymbol{\beta}_k(t_d^{(\zeta(s))})$ in (38) by substituting the optimal value for $\boldsymbol{\delta}_k$ in the linear functions in (40). Subsequently, we can recursively reconstruct the optimal values of the minimizers $\boldsymbol{\beta}_k(t_d^{(j)})$ in (35) by substituting the optimal value for $\boldsymbol{\beta}_k(t_d^{(\zeta(s))})$ and $\boldsymbol{\delta}_k$ in the linear functions in (39). Once optimal values of the optimization variables are determined, we may use the equations in (26) to update the corresponding state and parameter values for the next iteration.

An overview of the resulting algorithm is given by Algorithm 2. It is noted that the for loops in Lines 3 to 10 and in Lines 22 to 27 can be parallelized. The same remarks regarding the robustness and convergence of the algorithm apply as for Algorithm 1 in Section III.

Algorithm 2 Levenberg–Marquardt Algorithm With Parallelization

Input: $\{\mathbf{x}_0\}$, $\mathbf{a}_0$, λ_0, (parameters: σ, ρ_1, ρ_2)
Output: $\{\mathbf{x}_k\}$, $\mathbf{a}_k$

1: $k \leftarrow 0$
2: **while** $\left\| 2\left(\frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_k)\right)^T \mathbf{g}(\mathbf{b}_k) \right\| > \sigma$ **do**
3: **for** $s = 1$ to N_s **do**
4: $H_{s,1,k} \leftarrow q_{\zeta(s),k}$
5: **for** $j = 1$ to $\zeta(s+1) - \zeta(s) - 1$ **do**
6: Compute $l_{\zeta(s)+j,k}$ in (39)
7: Compute $H_{s,j+1,k}$ in (35)
8: **end for**
9: $w_{s,k} \leftarrow H_{s,\zeta(s+1)-\zeta(s),k}$
10: **end for**
11: $G_{1,k} \leftarrow w_{1,k}$
12: **for** $s = 2$ to N_s **do**
13: Compute $l_{\zeta(s),k}$ in (40)
14: Compute $G_{s,k}$ in (38)
15: **end for**
16: Compute minimizer $\boldsymbol{\delta}_k$ in (37)
17: $\mathbf{a}'_{k+1} \leftarrow \mathbf{a}_k + \boldsymbol{\delta}_k$
18: **for** $s = N_s$ to 2 **do**
19: Compute $\boldsymbol{\beta}_k(t_d^{(\zeta(s))})$ by evaluating $l_{\zeta(s),k}$ in (40)
20: $\mathbf{x}'_{k+1}(t_d^{(\zeta(s))}) \leftarrow \mathbf{x}_k(t_d^{(\zeta(s))}) + \boldsymbol{\beta}_k(t_d^{(\zeta(s))})$
21: **end for**
22: **for** $s = 1$ to N_s **do**
23: **for** $j = \zeta(s+1) - 1$ to $\zeta(s) + 1$ **do**
24: Compute $\boldsymbol{\beta}_k(t_d^{(j)})$ by evaluating $l_{j,k}$ in (39)
25: $\mathbf{x}'_{k+1}(t_d^{(j)}) \leftarrow \mathbf{x}_k(t_d^{(j)}) + \boldsymbol{\beta}_k(t_d^{(j)})$
26: **end for**
27: **end for**
28: **if** $\|\mathbf{g}(\mathbf{b}'_{k+1})\|^2 < \|\mathbf{g}(\mathbf{b}_k)\|^2$ **then**
29: $\mathbf{b}_{k+1} \leftarrow \mathbf{b}'_{k+1}$
30: $\lambda_{k+1} \leftarrow \frac{\lambda_k}{\rho_1}$
31: **else**
32: $\mathbf{b}_{k+1} \leftarrow \mathbf{b}_k$
33: $\lambda_{k+1} \leftarrow \rho_2 \lambda_k$
34: **end if**
35: $k \leftarrow k + 1$
36: **end while**

V. Network Sparsification

Consider the system model in (1)–(2), where the error of the first-principle model is modeled by an artificial neural network. To obtain a lean network, we propose to prune the network by sequentially removing network edges. By removing a network edge, we also remove the weight that corresponds to the network edge. Therefore, the number of parameters of the network decreases. Moreover, if all edges of a neuron are removed, then the neuron is no longer part of the network. Hence, removing network edges may also decrease the number of neurons in the network. We aim to only remove the network edges that have little or no effect on the minimal value of the cost function. In that way, we can maintain the quality of the fit after an edge removal. In general, we do not know by how much the minimal value of the cost function changes if any network edges are removed. To find out the difference in minimal value, we need to retrain the network after every change in network structure. Due to the large number of edges and the substantial computational effort it takes to retrain the network, it is often infeasible to retrain the network for every network configuration. In the following section, we present a method to identify which network edges are likely to have the least effect on the minimal value of the cost function. These edges are removed sequentially after verification. This

allows us to efficiently identify irrelevant network edges while keeping the overall computational cost relatively low.

A. Estimating the Change in Minimal Cost Function Value Due to a Network Edge Removal

We note that we can effectively remove a network edge by setting its corresponding weight to zero. Because the weights of the network are part of the parameter vector $\mathbf{a}$, determining the minimal value of the cost function after an edge removal is equivalent to determining the minimal cost function value after a certain element of the parameter vector $\mathbf{a}$ is set to zero. Suppose the artificial neural network in (2) is trained using Algorithm 2 of Section IV. Let the optimal state and parameter values be denoted by $\{\mathbf{x}_c\}$ and $\mathbf{a}_c$, or by $\mathbf{b}_c$, for short; see (15). Moreover, let the associated cost function value be given by $V_c = \|\mathbf{g}(\mathbf{b}_c)\|^2$; see (14). Similar to (20), we may approximate the output value of $\mathbf{g}(\mathbf{b})$ at any point $\mathbf{b} = \mathbf{b}_e$ close to $\mathbf{b}_c$ by the first-order Taylor series approximation

$$\mathbf{g}(\mathbf{b}_e) \approx \mathbf{g}(\mathbf{b}_c) + \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_c)\boldsymbol{\gamma}_e \tag{41}$$

where $\boldsymbol{\gamma}_e$ is defined as

$$\boldsymbol{\gamma}_e = \mathbf{b}_e - \mathbf{b}_c. \tag{42}$$

For ease of notation, we define

$$\boldsymbol{\beta}_e\left(t_d^{(j)}\right) = \mathbf{x}_e\left(t_d^{(j)}\right) - \mathbf{x}_c\left(t_d^{(j)}\right), \quad \boldsymbol{\delta}_e = \mathbf{a}_e - \mathbf{a}_c \tag{43}$$

for all $j \in \mathcal{J}_m$, such that $\boldsymbol{\gamma}_e$ in (42) is given by

$$\boldsymbol{\gamma}_e = \left[\boldsymbol{\beta}_e^T\left(t_d^{(1)}\right),\ \boldsymbol{\beta}_e^T\left(t_d^{(2)}\right),\ \ldots,\ \boldsymbol{\beta}_e^T\left(t_d^{(N_d)}\right),\ \boldsymbol{\delta}_e^T\right]^T. \tag{44}$$

Without loss of generality, we assume that the desired edge is removed from the network by setting the last element of the parameter vector $\mathbf{a} = \mathbf{a}_e$ to zero. Let the last element of $\mathbf{a}_c$ be denoted by α_c. If the last element of $\mathbf{a}_e$ is zero and the last element of $\mathbf{a}_c$ is α_c, it follows from (43) that the last element of $\boldsymbol{\delta}_e$ is $-\alpha_c$. Equivalently, the last element of $\boldsymbol{\gamma}_e$ is $-\alpha_c$; see (44). We let

$$\boldsymbol{\delta}_e = \begin{bmatrix} \hat{\boldsymbol{\delta}}_e \\ -\alpha_c \end{bmatrix}, \quad \boldsymbol{\gamma}_e = \begin{bmatrix} \hat{\boldsymbol{\gamma}}_e \\ -\alpha_c \end{bmatrix} \tag{45}$$

where $\hat{\boldsymbol{\delta}}_e$ and $\hat{\boldsymbol{\gamma}}_e$ are vectors containing all remaining elements of $\boldsymbol{\delta}_e$ and $\boldsymbol{\gamma}_e$, respectively. Assuming that $\mathbf{b}_e$ is close to $\mathbf{b}_c$ (i.e., assuming that $\boldsymbol{\gamma}_e$ is small), it follows that the minimal value of the cost function after removing the network edge can be accurately approximated by

$$V_e = \min_{\hat{\boldsymbol{\gamma}}_e} \left\| \mathbf{g}(\mathbf{b}_c) + \frac{d\mathbf{g}}{d\mathbf{b}}(\mathbf{b}_c)\boldsymbol{\gamma}_e \right\|^2. \tag{46}$$

This optimization problem is very similar to the one in (23). Thus, a similar approach to computing the solution can be applied.

As in Section IV, we may rewrite the optimization problem in (46) in the following recursive form:

$$V_e = \min_{\hat{\boldsymbol{\delta}}_e} \left\{ r_e + G_{N_s,e} \right\} \tag{47}$$

where $G_{N_s,e}$ is recursively defined as

$$G_{s,e} = \min_{\boldsymbol{\beta}_e\left(t_d^{(\zeta(s))}\right)} \left\{ w_{s,e} + G_{s-1,e} \right\} \tag{48}$$

for all $s \in \{2, 3, \ldots, N_s\}$, with $G_{1,e} = w_{1,e}$. For $s \in \{1, 2, \ldots, N_s\}$, the functions $w_{s,e}$ are given by

$$w_{s,e} = H_{s,\zeta(s+1)-\zeta(s),e} \tag{49}$$

where $H_{s,j+1,e}$ is obtained by the recursion

$$H_{s,j+1,e} = \min_{\boldsymbol{\beta}_e\left(t_d^{(\zeta(s)+j)}\right)} \left\{ q_{\zeta(s)+j,e} + H_{s,j,e} \right\} \tag{50}$$

for all $j \in \{1, 2, \ldots, \zeta(s+1) - \zeta(s) - 1\}$, with $H_{s,1,e} = q_{\zeta(s),e}$. Here, for all $j \in \mathcal{J}_d$, the functions $q_{j,e}$ and r_e are similarly defined as the functions $q_{j,k}$ and r_k in (29)–(31). The differences are that their arguments are related to the vector $\boldsymbol{\gamma}_e$ in (42) instead of the vector $\boldsymbol{\gamma}_k$ in (21), that the functions $\mathbf{p}_{\mathbf{x},j}$ and $\mathbf{p}_{\mathbf{a}}$, that are used in their definitions, have arguments related to $\mathbf{b}_c$ instead of $\mathbf{b}_k$, and that $\lambda_k = 0$.

In addition, we may compute the state and parameter values that correspond to the minimizer of the optimization problem in (46). These can serve as initial conditions for retraining the network if removing the network edge appears beneficial. Therefore, we introduce the linear functions that describe the minimizers of the optimization subproblems in (48) and (50), which can be used to evaluate the minimizer of the original optimization problem in (46); see Section IV for more details. Similar to (39), for any $s \in \{1, 2, \ldots, N_s\}$ and any integer j that satisfies $\zeta(s) < j < \zeta(s+1)$, the minimizer of the subproblem in (50) is given by

$$\boldsymbol{\beta}_e\left(t_d^{(j)}\right) = \begin{cases} l_{j,e}\left(\boldsymbol{\beta}_e\left(t_d^{(j+1)}\right), \boldsymbol{\delta}_e\right), & \text{if } s = 1 \\ l_{j,e}\left(\boldsymbol{\beta}_e\left(t_d^{(\zeta(N_s))}\right), \boldsymbol{\delta}_e\right), & \text{if } j = N_d \\ l_{j,e}\left(\boldsymbol{\beta}_e\left(t_d^{(\zeta(s))}\right), \boldsymbol{\beta}_e\left(t_d^{(j+1)}\right), \boldsymbol{\delta}_e\right), & \text{otherwise} \end{cases} \tag{51}$$

for some linear function $l_{j,e}$. Also, as in (40), for any $s \in \{2, 3, \ldots, N_s\}$, the minimizer of the subproblem in (38) can be written as

$$\boldsymbol{\beta}_e\left(t_d^{(\zeta(s))}\right) = \begin{cases} l_{\zeta(s),e}(\boldsymbol{\delta}_e), & \text{if } s = N_s \\ l_{\zeta(s),e}\left(\boldsymbol{\beta}_e\left(t_d^{(\zeta(s+1))}\right), \boldsymbol{\delta}_e\right), & \text{otherwise} \end{cases} \tag{52}$$

for some linear function $l_{\zeta(s),e}$.

The proposed method for estimating the minimal value of the cost function and the corresponding state and parameter values is summarized in Algorithm 3. Similar to Algorithm 2, note that the for loops in Lines 1 to 8 and in Lines 22 to 27 can be parallelized. Moreover, it should be noted that, if we want to compute the minimal cost function value for any other network edge removal instead of the one we have already computed, the results of Lines 1 to 13 of Algorithm 3 are identical and may be reused. This makes it relatively fast to compute all possible edge removals. Although Algorithm 3 may be applied to predict the minimal cost function value for any possible edge removal, in some cases, it is desirable

Algorithm 3 Estimating the Change in Minimal Cost Function Value Due to Network Edge Removal

Input: $\{\mathbf{x}_c\}$, $\mathbf{a}_c$
Output: V_e, $\{\mathbf{x}_e\}$, $\mathbf{a}_e$
for $s = 1$ to N_s **do**
  $H_{s,1,e} \leftarrow q_{\zeta(s),e}$
  for $j = 1$ to $\zeta(s+1) - \zeta(s) - 1$ **do**
    Compute $l_{\zeta(s)+j,e}$ in (51)
    Compute $H_{s,j+1,e}$ in (50)
  end for
  $w_{s,e} \leftarrow H_{s,\zeta(s+1)-\zeta(s),e}$
end for
$G_{1,e} \leftarrow w_{1,e}$
for $s = 2$ to N_s **do**
  Compute $l_{\zeta(s),e}$ in (52)
  Compute $G_{s,e}$ in (48)
end for
Compute minimizer $\hat{\boldsymbol{\delta}}_e$ in (47)
Compute V_e in (47)
$\boldsymbol{\delta}_e \leftarrow \begin{bmatrix} \hat{\boldsymbol{\delta}}_e \\ -\alpha_c \end{bmatrix}$
$\mathbf{a}_e \leftarrow \mathbf{a}_c + \boldsymbol{\delta}_e$
for $s = N_s$ to 2 **do**
  Compute $\boldsymbol{\beta}_e(t_d^{(\zeta(s))})$ by evaluating $l_{\zeta(s),e}$ in (52)
  $\mathbf{x}_e(t_d^{(\zeta(s))}) \leftarrow \mathbf{x}_c(t_d^{(\zeta(s))}) + \boldsymbol{\beta}_e(t_d^{(\zeta(s))})$
end for
for $s = 1$ to N_s **do**
  for $j = \zeta(s+1) - 1$ to $\zeta(s) + 1$ **do**
    Compute $\boldsymbol{\beta}_e(t_d^{(j)})$ by evaluating $l_{j,e}$ in (51)
    $\mathbf{x}_e(t_d^{(j)}) \leftarrow \mathbf{x}_c(t_d^{(j)}) + \boldsymbol{\beta}_e(t_d^{(j)})$
  end for
end for

to consider only a subset of network edges for removal. For example, by prioritizing the removal of edges related to monomials with a high degree, the polynomial output of the network may be of lower degree than without prioritization. In turn, this may improve the extrapolatory properties of the network model outside the measured region of the state. An example of this approach is given in Section VI-A.

B. Acceptance Criteria for Network Edge Removal

After we have estimated which edge removal results in the smallest increase in minimal cost function value, we compute the corresponding true minimal cost function value by retraining the network without the specific edge. We may use an acceptance criterion to decide whether to accept or reject a proposed removal of a network edge. The simplest criterion is solely based on the value of the cost function. A proposed removal of a network edge is accepted only if the value of the cost function after retraining the network with the proposed removal is lower than a preset limit value. Depending on the amount of initial overfitting, one may be able to remove many edges before exceeding the limit value after retraining the network, even if the limit value is only slightly larger than the minimal value of the cost function for the fully connected network.

Alternatively, we may use an information criterion to assess the quality of the estimate of the model error. In this case, the removal of a certain network edge is accepted only if the quality of the estimate does not decrease. Using an information criterion, both the value of the cost function as well as the number of network parameters are taken into account. There is a large selection of different information criteria to choose from. Several of them are described in detail in [3] and [10]. The choice of information criterion may significantly influence the sparsification process due to the differences in assumptions on which the various information criteria are derived. A similar approach of quality assessment has been proposed in [17], where Akaike's Information Criterion and the Bayesian Information Criterion are used to evaluate the results of the sparse identification approach in [2].

A third approach of arriving at an acceptance criterion is by cross-validation. Although there are many different forms of cross-validation, the main principle behind cross-validation is that the measurement data is partitioned in a training set, that is solely used for training the neural network, and a validation set (and/or test set; see [28], for example), to evaluate how well the predictions of the trained network generalize to previously unseen data. In that case, a proposed removal of a network edge is only accepted if it results in a lower value of the cost function based on the validation data. It is noted that determining the cost function value for the validation set requires solving the optimization problem in (6), where the (fixed) values of the network parameters are determined in the training stage. The use of a validation set avoids the problem of selecting a suitable limit value for the cost function or a suitable information criterion. However, because the training set and validation set must each cover the relevant part of the state space, this may put an unreasonably large demand on the amount of available measurement data.

VI. Examples

As discussed in Section I, our formulation of the system identification problem in Section II and the corresponding solution method in Sections IV and V allow for significantly lower data requirements and less stringent limitation on the network structure than other methods (e.g., in [2] and [29]) that are developed to construct a sparse system model. We illustrate our proposed sparse identification approach with two examples that could not have been computed with the methods in [2] and [29]. In Section VI-A, we identify the structure of an unknown system without measuring the full state vector, where the identification is based on measurements with different sampling rates. In Section VI-B, we train a network to approximate the error in the first-principle model without the requirement that the network model is linear in its parameters. We show that sparsification can simplify a network model while at the same time improve its accuracy by reducing overfitting.

A. Lorenz System

Consider the Lorenz system

$$\begin{aligned} \dot{x}_1(t) &= \sigma(x_2(t) - x_1(t)) \\ \dot{x}_2(t) &= x_1(t)(\rho - x_3(t)) - x_2(t) \\ \dot{x}_3(t) &= -\beta x_3(t) + x_1(t)x_2(t) \end{aligned} \tag{53}$$

with state $\mathbf{x}(t) = [x_1(t),\, x_2(t),\, x_3(t)]^T \in \mathbb{R}^3$ and parameters $\sigma = 10$, $\beta = (8/3)$ and $\rho = 28$. Without any information about the dynamics of the system, the first-principle model assumes a constant state

$$\mathbf{f}_{\text{phys}}(\mathbf{x}(t)) = \mathbf{0}. \tag{54}$$

This means that the true model error is given by

$$\mathbf{e}(t) = \begin{bmatrix} \sigma\,(x_2(t) - x_1(t)) \\ x_1(t)(\rho - x_3(t)) - x_2(t) \\ -\beta x_3(t) + x_1(t)x_2(t) \end{bmatrix}. \tag{55}$$

We use noisy measurements of the state vector to estimate the model error. The measurements of the first two elements of the state vector are taken every 0.3 time units from time $t = 0$ to time $t = 4.8$. The measurement of the third element of the state vector is taken every 0.4 time units on the same time interval. The measurement noise has a Gaussian distribution with zero mean and a variance of one. We use a polynomial network with monomials up to degree two to estimate the model error. A polynomial network is a network with a single hidden layer with monomials as neurons. Each output of the network is a linear combination of monomials. The corresponding monomial gains (i.e., the weights of the edges between the hidden layer and the output layer) are tuned to best match the measurement data. It is noted that all elements of the error vector (55) are polynomials of maximal degree two. Therefore, it is possible to obtain an exact expression of the model error using the network. We apply the sparsification approach presented in this article to identify the model error, with weighting matrices $\mathbf{W_x} = 10\mathbf{I}$ and $\mathbf{W_y} = \mathbf{I}$ and regularization constants $\mu_{\mathbf{x}} = 10^{-8}$ and $\mu_{\mathbf{a}} = 10^{-3}$; see Section II. The maximal discretization step size is set to $\Delta t = 10^{-3}$. We use the stage-wise sparsification approach mentioned in Section V-A, where we remove all irrelevant edges connected to monomials with the highest degree in the first stage before we continue to remove irrelevant edges connected to monomials with the second-highest degree in the second stage, etc. Following this approach, we obtain the following network model:

$$\mathbf{f}_{\text{net}}(\mathbf{x}(t)) = \begin{bmatrix} -9.378\; x_1(t) + 9.483\; x_2(t) \\ 28.779\; x_1(t) - 1.167\; x_2(t) - 1.024\; x_1(t)x_3(t) \\ -2.628\; x_3(t) + 0.960\; x_1(t)x_2(t) \end{bmatrix}. \tag{56}$$

Although we only have a few noisy measurements with a heterogeneous sampling rate at our disposal, the identified network model in (56) has the same structure as the model error in (55). Moreover, the identified parameter values are reasonably accurate. The time signals of the state, their estimates, and corresponding measurements are depicted in Fig. 1.

Now, suppose that only measurements of the first two state variables are available (i.e., $\mathbf{y}(t_m) \approx [x_1(t_m),\, x_2(t_m)]^T$). To successfully identify the model error by inferring the dynamics of the third state from the measurements of the first two states, the measurement rate should be increased and the variance of the measurement should be decreased. Let the measurements be taken every 0.01 time units from time $t = 0$ to time $t = 5$. Moreover, let the noise variance be given by 10^{-4}. We obtain the following network model:

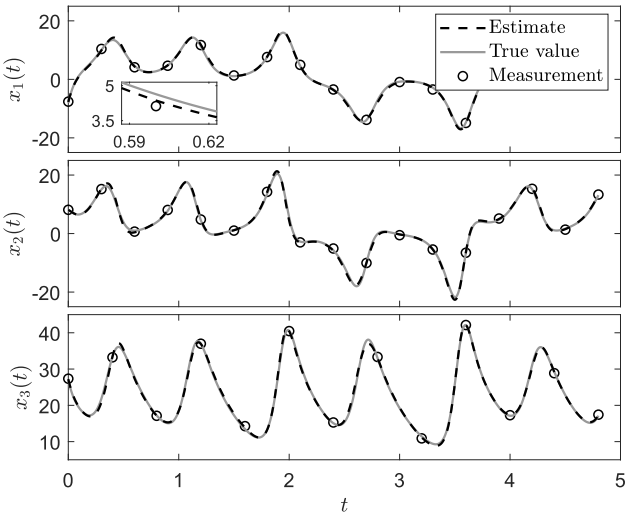

Fig. 1. Time signals of the state in (53) and the corresponding estimates using three state measurements with different sampling rates.

$$\mathbf{f}_{\text{net}}(\mathbf{x}(t)) = \begin{bmatrix} -9.997\; x_1(t) + 9.998\; x_2(t) \\ -0.960\; x_2(t) + 13.645\; x_1(t)x_3(t) \\ 5.418 - 2.643\; x_3(t) - 0.073\; x_1(t)x_2(t) \end{bmatrix}. \tag{57}$$

There seem to be significant differences in structure and value between the error vector in (55) and the network model in (57) at first glance. However, using the change of coordinate

$$\tilde{x}_3(t) = \gamma\,(\rho - x_3(t)) \tag{58}$$

with scaling parameter $\gamma \in \mathbb{R} \setminus \{0\}$, we get the following equivalent form of the dynamics in (53):

$$\begin{aligned} \dot{x}_1(t) &= \sigma\,(x_2(t) - x_1(t)) \\ \dot{x}_2(t) &= \frac{1}{\gamma} x_1(t)\tilde{x}_3(t) - x_2(t) \\ \dot{\tilde{x}}_3(t) &= \beta(\gamma\,\rho - \tilde{x}_3(t)) - \gamma\, x_1(t)x_2(t). \end{aligned} \tag{59}$$

For $\gamma = 0.073$, the corresponding model error is given by

$$\tilde{\mathbf{e}}(t) \approx \begin{bmatrix} -10\; x_1(t) + 10\; x_2(t) \\ -x_2(t) + 13.699\; x_1(t)x_3(t) \\ 5.451 - 2.667\; x_3(t) - 0.073\; x_1(t)x_2(t) \end{bmatrix}. \tag{60}$$

It is noted that the network model in (57) is an accurate estimate of the error vector in (60). The time signals of the equivalent dynamics in (59) and the corresponding estimates are depicted in Fig. 2. We observe that the differences between the true values of the states and the obtained estimates are small, also for the third state, for which there are no measurements available.

B. Van Der Pol Oscillator

Consider the forced Van der Pol oscillator

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) \\ \dot{x}_1(t) &= A\sin(\omega t) - x_1(t) + \mu\big(1 - x_1^2(t)\big)x_2(t) \end{aligned} \tag{61}$$

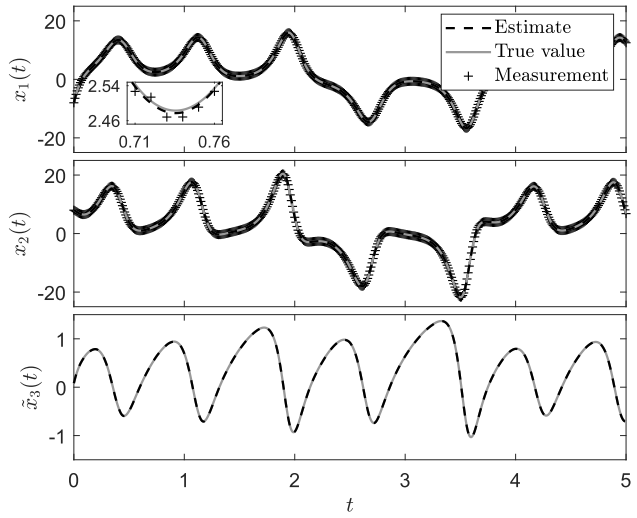

Fig. 2. Time signals of the state in (59) and the corresponding estimates using only the first two state measurements.

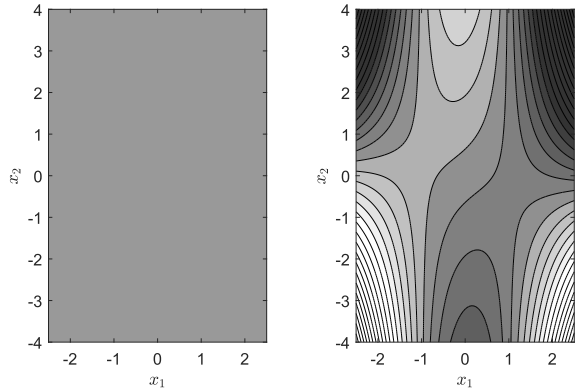

Fig. 3. Contour plots of the values of the first (left) and second (right) element of the error vector in (63). Note that the first element of the error vector is zero everywhere. Therefore, there are no contours to display. The gray color in the left plot corresponds to a value of zero. Lighter colors in the right plot indicate positive values; darker colors indicate negative values. The values in the right plot range from -23.5 to 23.5.

with state $\mathbf{x}(t) = [x_1(t),\ x_2(t)]^T \in \mathbb{R}^2$ and parameters $A = 1$, $\mu = 1$ and $\omega = 0.2$. Let the first-principle model and the corresponding model error respectively be given by

$$\mathbf{f}_{\text{phys}}(t, \mathbf{x}(t)) = \begin{bmatrix} x_2(t) \\ A\sin(\omega t) \end{bmatrix} \tag{62}$$

and

$$\mathbf{e}(t) = \begin{bmatrix} 0 \\ -x_1(t) + \mu\big(1 - x_1^2(t)\big)x_2(t) \end{bmatrix}. \tag{63}$$

The error in the first-principle model is visualized in Fig. 3. It is modeled by an artificial neural network with two hidden layers of ten units each. The (differentiable) exponential linear unit with shape parameter $a = 1$ (see [5]) is used as activation function for all neurons. The inputs of the network are the states x_1 and x_2. It is noted that, if the value of A would be uncertain, we could include $\sin(\omega t)$ as an extra input to the network [i.e., $u(t) = \sin(\omega t)$ in (2)] to also learn the model error related to an incorrect estimate of A. The outputs of the network are the functions $f_{\text{net},1}(\mathbf{x}(t))$ and $f_{\text{net},2}(\mathbf{x}(t))$ that represent the estimates of the two elements of the error vector.

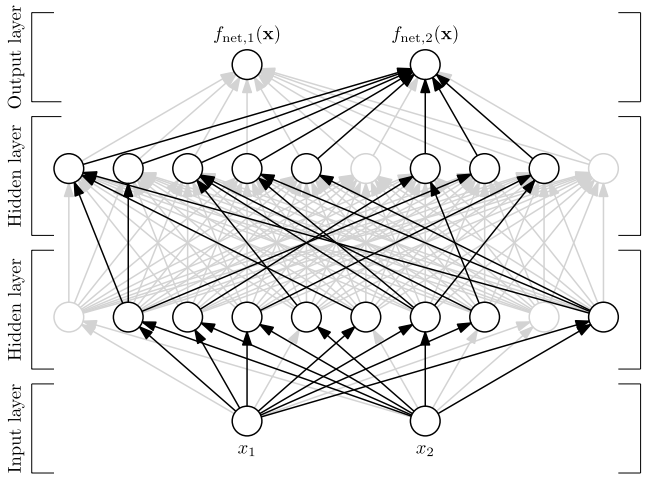

Fig. 4. Network layout before sparsification (gray) and after sparsification (black).

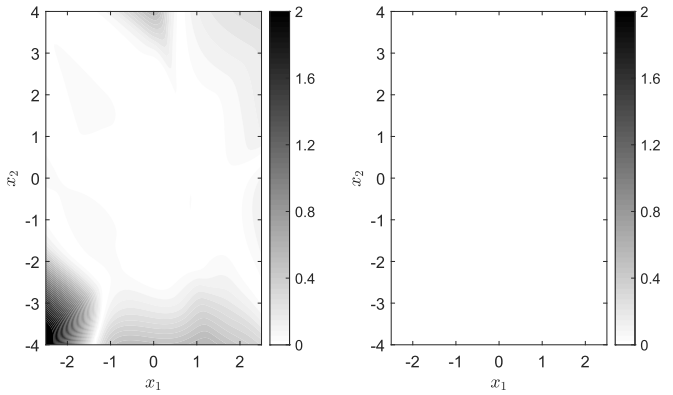

Fig. 5. Absolute value of the estimation error for the first element of the error vector in (63) for the fully connected network (left) and the sparsified network (right). A value of zero is indicated in white. A value of two or higher is indicated in black.

Because there exist no network parameters that represent the model error exactly for all values of $\mathbf{x}$, the network provides only a local approximation of the model error. The network is trained based on noisy measurements of the (full) state of the system. The measurements are taken from time $t = 0$ to time $t = 100$, with a step size of 0.1 between subsequent measurements. The measurement noise has a Gaussian distribution with a zero mean and a variance of 0.01.

By applying the sparsification approach in Section V with weighting matrices $\mathbf{W_x} = \mathbf{W_y} = \mathbf{I}$, regularization constants $\mu_{\mathbf{x}} = 0$ and $\mu_{\mathbf{a}} = 10^{-3}$, and maximal discretization step size $\Delta t = 10^{-3}$, we are able to reduce the number of network edges from 140 to 36 and the number of neurons from 20 to 16. This means that the overall number of network parameters (i.e., weights and biases) is decreased from 160 to 52. The layout of the sparsified network is depicted in Fig. 4. The absolute values of the estimation errors for the two elements of the error vector in (63) for the fully connected network as well as the sparsified network are depicted in Figs. 5 and 6. The state values that correspond to the process measurements are visualized by black dots in Fig. 7. Despite having much fewer edges and parameters, we observe that the sparsified network arguably represents the error in the first-principle model better on average in a neighborhood around the process

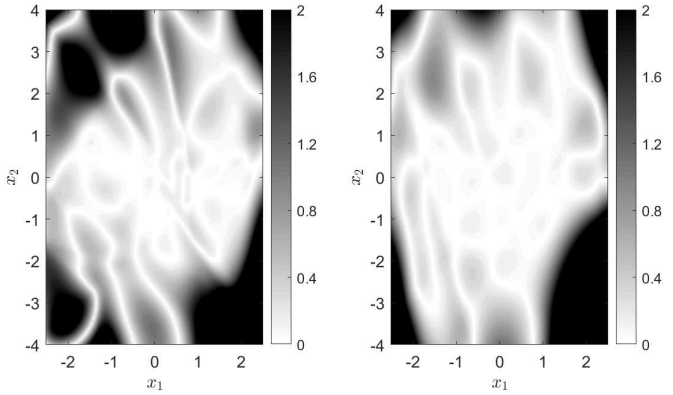

Fig. 6. Absolute value of the estimation error for the second element of the error vector in (63) for the fully connected network (left) and the sparsified network (right). A value of zero is indicated in white. A value of two or higher is indicated in black.

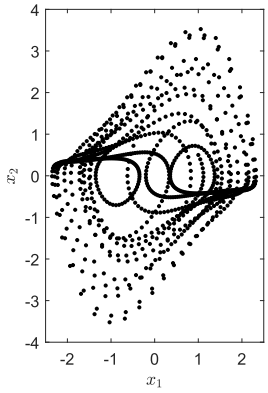

Fig. 7. Location of the process measurements in the state space.

measurements than the fully connected network does. The sparsified network captures the first element of the error vector more accurately than the fully connected network for 100% of the area of the state space shown in the figures; it captures the second element of the error vector more accurately for roughly 63% of the area of the state space shown in the figures. This improvement of accuracy can be largely attributed to the reduction of overfitting due to network sparsification.

VII. Discussion

In this article, we have presented an efficient method for sparse identification of dynamical systems described by ordinary differential equations. We have shown that the Levenberg–Marquardt algorithm, on which our method is based, can be rewritten in a form that enables parallel computation to a large extent. Therefore, the wall time required to solve the identification problem can be greatly decreased. In addition, we have presented a sparsification approach based on backward elimination that utilizes the same time-efficient computation strategy to estimate the relevance of any given network edge, so that only those network edges that have a small effect on the quality of the model fit are removed. We have illustrated with two examples that our method is not affected by many of the limitations, such as fixed sampling rates, full state measurements, and linearity of the model, that plague other methods (e.g., in [2] and [29]). This flexibility is essential for the applicability of the method in many industrial settings. The main drawback of our approach is the relatively large computational demand. However, utilizing the time-efficient algorithm presented in this article, the wide applicability of our method is likely to outweigh the larger computational cost in many practical scenarios.

Appendix A
Algorithm for Generating Discretization Points

It is noted that all continuous-time signals in (6) are defined on the interval $[\underline{t}_m, \overline{t}_m]$. For each subinterval $[t_m^{(i)}, t_m^{(i-1)}]$, we define the number of discretization points between subsequent measurements

$$N_r^{(i)} = \left\lceil \frac{t_m^{(i+1)} - t_m^{(i)}}{\Delta t} \right\rceil \tag{64}$$

and the set of positive integers

$$\mathcal{J}_r^{(i)} = \left\{ j \in \mathbb{N} : 1 \leq j \leq N_r^{(i)} \right\} \tag{65}$$

for all $i \in \mathcal{I}_m \setminus \{N_m\}$. With these definitions in place, let the discretization points between subsequent measurements be given by

$$\mathcal{T}_r^{(i)} = \left\{ t_d \in \mathbb{R} : \left(\exists j \in \mathcal{J}_r^{(i)}\right) \left[t_d = t_m^{(i)} + \frac{t_m^{(i+1)} - t_m^{(i)}}{N_r^{(i)}} (j-1) \right] \right\} \tag{66}$$

for all $i \in \mathcal{I}_m \setminus \{N_m\}$. Additionally, let $\mathcal{T}_r^{(N_m)} = \{t_m^{(N_m)}\}$. The set of all discretization points is given by

$$\mathcal{T}_d = \bigcup_{i \in \mathcal{I}_m} \mathcal{T}_r^{(i)}. \tag{67}$$

It can be noted that $\mathcal{T}_m \subseteq \mathcal{T}_d$. The cardinality of $\mathcal{T}_d$ is given by

$$N_d = \sum_{i \in \mathcal{I}_m} N_r^{(i)} \tag{68}$$

where $N_r^{(N_m)} = 1$.

Appendix B
Algorithm for Solving the Optimization Subproblems

It is noted that the optimization subproblems in (35), (37), and (38) can all be written in the form

$$\min_{\mathbf{r}} \{ g_1(\mathbf{r}, \mathbf{s}) + g_2(\mathbf{r}, \mathbf{s}) \} \tag{69}$$

where g_1 and g_2 are quadratic, sum-of-squares functions of the optimization variables we want to minimize over $\mathbf{r}$, as well as other optimization variables $\mathbf{s}$. Because g_1 and g_2 are quadratic and sum of squares, they can be written as

$$g_i(\mathbf{r}, \mathbf{s}) = \left\| \begin{bmatrix} \mathbf{M}_{\mathbf{r},i} & \mathbf{M}_{\mathbf{s},i} & \mathbf{M}_{1,i} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{s} \\ 1 \end{bmatrix} \right\|^2 \tag{70}$$

for $i \in \{1, 2\}$ and some coefficient matrices $\mathbf{M}_{\mathbf{r},i}$, $\mathbf{M}_{\mathbf{s},i}$, and $\mathbf{M}_{1,i}$. It is noted that the sum of g_1 and g_2 can be written as

$$g_1(\mathbf{r}, \mathbf{s}) + g_2(\mathbf{r}, \mathbf{s}) = \left\| \begin{bmatrix} \mathbf{M}_{\mathbf{r}} & \mathbf{M}_{\mathbf{s}} & \mathbf{M}_1 \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{s} \\ 1 \end{bmatrix} \right\|^2 \tag{71}$$

with

$$\mathbf{M_r} = \begin{bmatrix} \mathbf{M_{r,1}} \\ \mathbf{M_{r,2}} \end{bmatrix}, \quad \mathbf{M_s} = \begin{bmatrix} \mathbf{M_{s,1}} \\ \mathbf{M_{s,2}} \end{bmatrix}, \quad \mathbf{M}_1 = \begin{bmatrix} \mathbf{M}_{1,1} \\ \mathbf{M}_{1,2} \end{bmatrix}. \tag{72}$$

Because the cost function $g_1 + g_2$ is quadratic and sum of squares, all values of $\mathbf{r}$ for which the gradient of the cost function with respect to $\mathbf{r}$ is zero are minimizers. That is, $\mathbf{r}$ minimizes the cost function $g_1 + g_2$ if it satisfies

$$2\mathbf{M}_\mathbf{r}^T(\mathbf{M_r r} + \mathbf{M_s s} + \mathbf{M}_1) = \mathbf{0}. \tag{73}$$

Therefore, a minimizer is given by

$$\mathbf{r} = -\left(\mathbf{M}_\mathbf{r}^T\mathbf{M_r}\right)^+\mathbf{M}_\mathbf{r}^T\begin{bmatrix}\mathbf{M_s} & \mathbf{M}_1\end{bmatrix}\begin{bmatrix}\mathbf{s} \\ 1\end{bmatrix}. \tag{74}$$

It is noted that this minimizer is a linear function of the other optimization variables $\mathbf{s}$. This minimizer is unique if $\mathbf{M}_\mathbf{r}^T\mathbf{M_r}$ is invertible, in which case we can exchange the pseudoinverse by the regular inverse. The corresponding solution of the optimization problem in (69) is given by

$$\min_\mathbf{r}\{g_1(\mathbf{r},\mathbf{s}) + g_2(\mathbf{r},\mathbf{s})\} = \left\|\mathbf{P_r}\begin{bmatrix}\mathbf{M_s} & \mathbf{M}_1\end{bmatrix}\begin{bmatrix}\mathbf{s} \\ 1\end{bmatrix}\right\|^2 \tag{75}$$

with

$$\mathbf{P_r} = \mathbf{I} - \mathbf{M_r}\left(\mathbf{M}_\mathbf{r}^T\mathbf{M_r}\right)^+\mathbf{M}_\mathbf{r}^T. \tag{76}$$

A. Computation Using the QR-Decomposition

The computation of the minimizer in (74) and solution in (75) of the optimization problem in (69) can be simplified using the QR-decomposition. The QR decomposition decomposes any real, rectangular matrix $\mathbf{A}$ into an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$ (where the rank of $\mathbf{R}$ is equal to its number of nonzero diagonal elements), such that $\mathbf{A} = \mathbf{QR}$; see [8], for example. It is noted that $\mathbf{A}^T\mathbf{A} = (\mathbf{QR})^T\mathbf{QR} = \mathbf{R}^T\mathbf{R}$, because $\mathbf{Q}$ is orthogonal. Thus, for any partitioned, real, rectangular matrix

$$\mathbf{A} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2\end{bmatrix} \tag{77}$$

we have

$$\begin{aligned}\mathbf{A}^T\mathbf{A} &= \begin{bmatrix}\mathbf{A}_1^T\mathbf{A}_1 & \mathbf{A}_1^T\mathbf{A}_2 \\ \mathbf{A}_2^T\mathbf{A}_1 & \mathbf{A}_2^T\mathbf{A}_2\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{R}_{11}^T\mathbf{R}_{11} & \mathbf{R}_{11}^T\mathbf{R}_{12} \\ \mathbf{R}_{12}^T\mathbf{R}_{11} & \mathbf{R}_{12}^T\mathbf{R}_{12} + \mathbf{R}_{22}^T\mathbf{R}_{22}\end{bmatrix} = \mathbf{R}^T\mathbf{R}\end{aligned} \tag{78}$$

for some partitioned, upper triangular matrix

$$\mathbf{R} = \begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{R}_{22}\end{bmatrix} \tag{79}$$

where the elements of $\mathbf{R}$ are implicitly given by (78). It follows that

$$\mathbf{R}_{11}^+\mathbf{R}_{12} = \left(\mathbf{R}_{11}^T\mathbf{R}_{11}\right)^+\mathbf{R}_{11}^T\mathbf{R}_{12} = \left(\mathbf{A}_1^T\mathbf{A}_1\right)^+\mathbf{A}_1^T\mathbf{A}_2 \tag{80}$$

and

$$\begin{aligned}\mathbf{R}_{22}^T\mathbf{R}_{22} &= \mathbf{A}_2^T\mathbf{A}_2 - \mathbf{R}_{12}^T\mathbf{R}_{12} \\ &= \mathbf{A}_2^T\mathbf{A}_2 - \mathbf{R}_{12}^T\mathbf{R}_{11}\left(\mathbf{R}_{11}^T\mathbf{R}_{11}\right)^+\mathbf{R}_{11}^T\mathbf{R}_{12} \\ &= \mathbf{A}_2^T\left(\mathbf{I} - \mathbf{A}_1\left(\mathbf{A}_1^T\mathbf{A}_1\right)^+\mathbf{A}_1^T\right)\mathbf{A}_2.\end{aligned} \tag{81}$$

Now, if $\mathbf{A} = \begin{bmatrix}\mathbf{M_r} & \mathbf{M_s} & \mathbf{M}_1\end{bmatrix}$, with $\mathbf{A}_1 = \mathbf{M_r}$ and $\mathbf{A}_2 = \begin{bmatrix}\mathbf{M_s} & \mathbf{M}_1\end{bmatrix}$, by comparison with (80) and (81), we obtain the following simple expressions for the minimizer in (74) and the solution in (75):

$$\mathbf{r} = -\mathbf{R}_{11}^+\mathbf{R}_{12}\begin{bmatrix}\mathbf{s} \\ 1\end{bmatrix} \tag{82}$$

and

$$\min_\mathbf{r}\{g_1(\mathbf{r},\mathbf{s}) + g_2(\mathbf{r},\mathbf{s})\} = \left\|\mathbf{R}_{22}\begin{bmatrix}\mathbf{s} \\ 1\end{bmatrix}\right\|^2. \tag{83}$$

Hence, the minimizer in (74) and the solution in (75) can be efficiently computed using the QR-decomposition.

REFERENCES

[1] R. E. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton Univ. Press, 1957.

[2] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 15, pp. 3932–3937, 2015.

[3] K. P. Burnham and D. R. Anderson, *Model Selection Multimodel Inference: A Practical Information-Theoretic Approach*, 2nd ed. New York, NY, USA: Springer, 2002.

[4] S. L. Campbell and C. D. Meyer, *Generalized Inverses of Linear Transformations*. Philadelphia, PA, USA: SIAM, 2009.

[5] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," 2015, *arXiv:1511.07289*.

[6] M. A. Efroymson, *Mathematical Methods for Digital Computers* (Multiple regression analysis). New York, NY, USA: Wiley, 1960, pp. 191–203.

[7] X. Hong, R. J. Mitchell, S. Chen, C. J. Harris, K. Li, and G. W. Irwin, "Model selection approaches for non-linear system identification: A review," *Int. J. Syst. Sci.*, vol. 39, no. 10, pp. 925–946, Oct. 2008.

[8] S. F. Hsieh, K. J. R. Liu, and K. Yao, "A unified square-root-free approach for QRD-based recursive-least-squares estimation," *IEEE Trans. Signal Process.*, vol. 41, no. 3, pp. 1405–1409, Mar. 1993.

[9] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 474, no. 2219, Nov. 2018, Art. no. 20180335.

[10] S. Konishi and G. Kitagawa, *Information Criteria and Statistical Modeling*. New York, NY, USA: Springer, 2008.

[11] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, pp. 987–1000, Sep. 1998.

[12] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proc. ICML*, 2011, pp. 265–272.

[13] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quart. J. Appl. Math.*, vol. 2, no. 2, pp. 164–168, Jul. 1944.

[14] Z. Li *et al.*, "Fourier neural operator for parametric partial differential equations," 2020, *arXiv:2010.08895*.

[15] L. Ljung, "Perspectives on system identification," *Annu. Rev. Control*, vol. 34, no. 1, pp. 1–12, 2010.

[16] K. Madsen, H. B. Nielsen, and O. Tingleff, "Methods for non-linear least squares problems," Dept. Inform. Math. Model., Tech. Univ. Denmark, Lyngby, Denmark, Apr. 2004.

[17] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor, "Model selection for dynamical systems via sparse regression and information criteria," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 473, no. 2204, Aug. 2017, Art. no. 20170009.

[18] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, Jun. 1963.

[19] A. Mauroy and J. Goncalves, "Koopman-based lifting techniques for nonlinear systems identification," *IEEE Trans. Autom. Control*, vol. 65, no. 6, pp. 2550–2565, Jun. 2020.

[20] A. Miller, *Subset Selection Regression*. Boca Raton, FL, USA: CRC Press, 2002.
[21] I. Mukherjee and S. Routroy, "Comparing the performance of neural networks developed by using Levenberg–Marquardt and Quasi-Newton with the gradient descent algorithm for modelling a multiple response grinding process," *Expert Syst. Appl.*, vol. 39, no. 3, pp. 2397–2407, Feb. 2012.
[22] J. Nduhura-Munga, G. Rodriguez-Verjan, S. Dauzere-Peres, C. Yugma, P. Vialletelle, and J. Pinaton, "A literature review on sampling techniques in semiconductor manufacturing," *IEEE Trans. Semicond. Manuf.*, vol. 26, no. 2, pp. 188–195, May 2013.
[23] S. Pan and K. Duraisamy, "Data-driven discovery of closure models," *SIAM J. Appl. Dyn. Syst.*, vol. 17, no. 4, pp. 2381–2413, Jan. 2018.
[24] J. Pujol, "The solution of nonlinear inverse problems and the Levenberg–Marquardt method," *Geophysics*, vol. 72, no. 4, pp. W1–W16, Jul. 2007.
[25] M. Quade, M. Abel, J. Nathan Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for rapid model recovery," *Chaos: Interdiscipl. J. Nonlinear Sci.*, vol. 28, no. 6, Jun. 2018, Art. no. 063116.
[26] M. Raissi and G. E. Karniadakis, "Hidden physics models: Machine learning of nonlinear partial differential equations," *J. Comput. Phys.*, vol. 357, pp. 125–141, Mar. 2018.
[27] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019.
[28] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[29] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Sci. Adv.*, vol. 3, no. 4, Apr. 2017, Art. no. e1602614.
[30] S. H. Rudy, J. Nathan Kutz, and S. L. Brunton, "Deep learning of dynamics and signal-noise decomposition with time-stepping constraints," *J. Comput. Phys.*, vol. 396, pp. 483–506, Nov. 2019.
[31] J. Sjöberg *et al.*, "Nonlinear black-box modeling in system identification: A unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995.
[32] H. H. Tan and K. H. Lim, "Review of second-order optimization techniques in artificial neural networks backpropagation," in *Proc. IOP Conf., Mater. Sci. Eng.*, vol. 495, 2019, Art. no. 012003.
[33] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Stat. Soc., B (Methodol.)*, vol. 58, no. 1, pp. 267–288, Jan. 1996.
[34] M. K. Transtrum and J. P. Sethna, "Improvements to the Levenberg–Marquardt algorithm for nonlinear least-squares minimization," 2012, *arXiv:1201.5885*.
[35] X. Ying, "An overview of overfitting and its solutions," in *Proc. J. Phys., Conf.*, vol. 1168, 2019, Art. no. 022022.
[36] P. Zheng, T. Askham, S. L. Brunton, J. N. Kutz, and A. Y. Aravkin, "A unified framework for sparse relaxed regularized regression: SR3," *IEEE Access*, vol. 7, pp. 1404–1423, 2019.

Mark Haring received the B.Sc. and M.Sc. degrees in mechanical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 2008 and 2011, respectively, and the Ph.D. degree in engineering cybernetics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2016.

He is currently a Research Scientist with the Department of Mathematics and Cybernetics, SINTEF Digital, Trondheim. His research interests include automatic control, adaptive control, machine learning, and estimation.

Esten Ingar Grøtli (Member, IEEE) received the M.Sc. and Ph.D. degrees in engineering cybernetics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2005 and 2010, respectively.

Since 2014, he has been working on several projects within robotics and process control with the Department of Mathematics and Cybernetics, SINTEF Digital, Trondheim, where he is currently a Senior Research Scientist. His research interests include estimation, machine learning, and sensor fusion.

Signe Riemer-Sørensen received the Ph.D. degree in astrophysics from the Dark Cosmology Center, Niels Bohr Institute, University of Copenhagen, Copenhagen, Denmark in 2009.

From 2010 to 2017, she held positions with the School of Mathematics and Physics, University of Queensland, Brisbane, Australia, and the Institute of Theoretical Astrophysics, University of Oslo, Oslo, Norway. Since 2018, she has been a Researcher with SINTEF Digital with a focus on the development of hybrid machine learning algorithms combining data-driven methods and domain knowledge, for use in industrial settings, in particular within the domains of energy, construction, and logistics.

Katrine Seel received the M.Sc. degree in marine cybernetics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2017.

She currently works with the Department of Engineering Cybernetics, NTNU, as a Ph.D. student, and holds a part-time position with SINTEF Digital, in the group for analytics and artificial intelligence. Her research interests include combining model predictive control with learning methods, such as reinforcement learning.

Kristian Gaustad Hanssen received the M.Sc. and Ph.D. degrees in engineering cybernetics from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2009 and 2017.

He is currently working as a Research Scientist with the Department of Mathematics and Cybernetics, SINTEF Digital, Trondheim, Norway. His research interests are within optimal control, stochastic programming, and data analysis.