Real-time parameter updating for nonlinear digital twins using inverse mapping models and transient-based features

Bas M. Kessels[1*], Rob H.B. Fey[1] and Nathan van de Wouw[1]

[1*]Mechanical Engineering, Eindhoven University of Technology, Den Dolech 2, Eindhoven, 5612 AZ, the Netherlands.

*Corresponding author(s). E-mail(s): b.m.kessels@tue.nl;
Contributing authors: r.h.b.fey@tue.nl; n.v.d.wouw@tue.nl;

Abstract

In the context of digital twins, it is essential that a model gives an accurate description of the (controlled) dynamic behavior of a physical system during the system's entire operational life. Therefore, model updating techniques are required that enable real-time updating of physically interpretable parameter values and are applicable to a wide range of (nonlinear) dynamical systems. As traditional, iterative, parameter updating methods may be computationally too expensive for real-time updating, the Inverse Mapping Parameter Updating (IMPU) method is proposed as an alternative. For this method, first, an Artificial Neural Network (ANN) is trained offline using novel features of simulated transient response data. Then, in the online phase, this ANN maps, with little computational cost, a set of measured output response features to parameter estimates enabling real-time model updating. In this paper, various types of transient response features are introduced to update parameter values of nonlinear dynamical systems with increased computational efficiency and accuracy. To analyze the efficacy of these features, the IMPU method is applied to a (simulated) nonlinear multibody system. It is shown that a smart selection of features, based on, e.g., the frequency content of the transient response, can improve the accuracy of the estimated parameter values, leading to more accurate updated models. Furthermore, the generalization capabilities of the ANNs are analyzed for these feature types, by varying the number of training samples and assessing the effect of incomplete training data. It is shown that the IMPU method can predict parameter values that are not part of the training data with acceptable accuracy as well.

Keywords: Model updating, Parameter estimation, Neural networks, Transient-based features, Digital twin, Nonlinear systems.

1 Introduction

As part of the fourth industrial revolution, digital twins hold the promise of increasing the efficiency, and lowering costs and throughput times of, among others, high-tech engineering systems and manufacturing processes [1], such as, e.g., wire bonder machines, lithography systems, or steel manufacturing plants. As the name 'digital twin' suggests, this potential is achieved by having a digital representation of a physical system.

Using this digital twin, the behavior of the 'physical twin' is predicted risk-free by simulating and analyzing the digital copy of the system [2]. In the design process of future generations of the system, this provides engineers with valuable

insight into the dynamics of the current system, enabling improved design of the next generation. Additionally, measured behavior of the actual system can be compared to behavior predicted by a digital twin of the healthy system for the benefit of Structural Health Monitoring (SHM) [3], enabling efficient maintenance management, minimizing maintenance cost, and leading to an increase of the lifetime of the system. Furthermore, during the operation of a system, a digital twin is able to suggest different operation modes or adapt mission planning if it senses that the current state of the system is not suited for an originally planned operation mode or mission [4]. This application can also be extended to using a dedicated digital twin for each machine in a fleet of akin machines, based on its most recent measured state, to account for machine-to-machine variations originating from, e.g., manufacturing tolerances. Availability of such dedicated digital twins allows for adapting model-based (feedforward) controllers to enhance the performance of each individual machine.

Coined in 2014 by Grieves [5], the concept of digital twins is relatively novel and still faces multiple broad challenges, such as managing heterogeneous models across varying disciplines and combining models and big-data for, among others, fault-detection and performance optimization. Another challenge is related to the ever-present mismatch between measurements of a physical system and corresponding predictions of its digital twin. For the digital twin to reach its full potential, this mismatch should be minimal throughout the system's operational life [1, 4]. This challenge of giving digital twin life-long learning capabilities is referred to as autonomous model updating and is considered in this paper.

In Mottershead et al. [6, 7], the aforementioned mismatch between measurement and model predictions is contributed to three distinct error sources; incorrect model structure [8, 9], discretization errors [10], and/or errors due to incorrect parameter values. This paper focuses on model parameter updating as to minimize the last error source. Therefore, model structure and discretization errors are assumed negligible.

Given the intended application in a general digital twin context, key requirements for an autonomous model updating procedure are defined as follows: 1) although control and SHM will not be considered in this paper yet, we aim to develop a model updating methodology applicable in the context of model-based control and/or SHM; therefore, updating should be performed in real-time, 2) parameter updates should be physically interpretable to enable insightful SHM, and 3) updating should be applicable to nonlinear dynamical models. The presentation of an autonomous model updating method simultaneously fulfilling these three requirements, is the main contribution of the current paper. The corresponding technical contributions will be discussed in more detail later in this section.

Approaches for model parameter updating found in literature are divided in three types. The first type concerns updating methods originating from the field of structural dynamics, see, e.g., [7, 11] for an overview. Although some exceptions exist, see [12–15], these updating methods are typically limited to linear(ized) systems [6, 16]. Furthermore, these methods are based on expensive iterative sensitivity methods [14, 15, 17–23] or physically non-interpretable direct methods [19, 24–26]. Consequently, these techniques do not meet the three requirements mentioned above and are therefore not regarded as a viable solution to the problem considered here.

Alternatively, system identification techniques are used to find a model that describes measured data. For an overview of system identification techniques see, for example, [27–29]. Note that, in contrast to model updating in the field of, typically linear(ized), structural dynamics, system identification has been developed for nonlinear systems [30, 31]. However, due to the fact that in system identification generic model classes are fitted to data, the identified models are difficult to interpret from a physical perspective [32], and thus do not satisfy the above requirements.

The third approach to model updating employs Machine Learning (ML). Here, again, we can distinguish between, firstly, identification, i.e., identifying a system without using a first-principles based reference model, and, secondly, updating (parameters of) a predefined reference model. An example of the former is the work of Li on identifying nonlinear normal modes using physics-integrated deep learning [33]. An other example of ML-based identification is the work

by Karniadakis et al. [34–36] on Physics-Informed Neural Networks (PINNs), also adopted by other researchers [37]. In this method, (partial) differential equations governing the measured system are learned using a neural network. The parameters of these (partial) differential equations, however, in general do not refer to directly identifiable physical quantities, e.g., masses, damping, and stiffness constants. A similar argument holds for the Sparse Identification of Nonlinear Dynamics (SINDy) algorithm developed by Brunton et al. [38–41]. An ML-based method more closely related to (physically interpretable) model updating is found in the work of Willcox et al. [42–46]. Here, a (first-principles) model is selected from a predefined library of models, representing systems with various fault types, using an optimal decision tree. Due to a prerequisite library of models, the 'updated' model is, however, limited to a discrete set of models that are manually defined by the user.

As an alternative, the inverse problem of estimating physically interpretable parameter values in a continuous domain using measurement data can be solved using an Inverse Mapping Model (IMM). Here, the IMM maps, with small computational costs, a set of measured features, describing the (dynamical) behavior of a system, to a set of corresponding parameter values of the reference model. In literature, (trained) Artificial Neural Networks (ANNs) have been found suitable to constitute these IMMs due to their generic function approximation characteristics [16, 47–50]. Consequently, this methodology was coined the Neural Network Updating Method (NNUM) by Atalla [16]. Note that, since other generic function approximation algorithms may also be used to constitute IMMs, the authors prefer to refer to this methodology as the Inverse Mapping Parameter Updating (IMPU) method. For example, Zimmerman uses genetic algorithms [51]. Moreover, in recent work of the authors, Gaussian processes are used as IMMs, with the added benefit of enabling uncertainty quantification for the estimated parameter values [52]. To the best of the authors' knowledge, this IMPU methodology has, however, only been applied to linear(ized) dynamical systems.

Therefore, the three main contributions of this work are:

1. Extension of the IMPU method towards nonlinear dynamical systems. Hereto, we propose to use transient-based output signal features to update nonlinear dynamical systems online by inferring (physically interpretable) parameter values using IMMs with little (online) computational cost. Here, multiple types of transient-based features are introduced that use engineering knowledge to increase accuracy and computational efficiency of the IMPU methodology.
2. An application and evaluation of the introduced methodology and feature types to a simulated nonlinear multi-body system.
3. An extensive evaluation of the generalization capabilities of the introduced feature types by varying the number of training samples and using incomplete training data.

In preliminary work of the authors [53], the IMPU method has been applied to update a nonlinear dynamical model. However, this work only discussed the use of a single, straightforward transient-based feature type (samples of measured time-series data as directly obtained from measurements). With respect to this preliminary work, the contributions of the current paper encompass the detailed description of the IMPU method for nonlinear systems, the definition of multiple distinct feature types (inspired by engineering knowledge), the application, evaluation, and comparison of these feature types on a simulated multi-body system, and the evaluation of their generalization capabilities.

The outline of this paper is as follows. In Section 2, a formal definition of the model parameter updating problem is given. Moreover, the usage of transient-based output features to update, online, interpretable parameter values of nonlinear models with low computational costs is introduced. Subsequently, in Section 3, different transient-based output features are introduced. In Section 4, the introduced technique is applied on a (simulated) academic demonstrator and results are analyzed. Additionally, this section analyzes the generalization capabilities of the aforementioned transient-based features. Finally, conclusions are discussed in Section 5.

2 Model updating using inverse mapping models

2.1 Problem definition

It is assumed that a nonlinear, first-principles reference model of a dynamical system is available. The dynamics of this model are given in first-order State Space (SS) form:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{g}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}), \\ \boldsymbol{y}(t) &= \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}). \end{aligned} \tag{1}$$

Here, $\boldsymbol{x} = \left[\boldsymbol{q}^\top, \dot{\boldsymbol{q}}^\top\right]^\top \in \mathbb{R}^{2n_{\text{DoF}}}$ represents the state vector of the dynamical system that generally consists of translation and/or rotation Degrees Of Freedom (DoFs), $\boldsymbol{q}$, and their corresponding time derivatives, $\dot{\boldsymbol{q}}$, where n_{DoF} represents the number of DoFs in the model. Furthermore, $\boldsymbol{y} \in \mathbb{R}^{n_{\text{out}}}$ is a vector containing n_{out} outputs and $\boldsymbol{u} \in \mathbb{R}^{n_{\text{in}}}$ contains n_{in} inputs. Finally, the parameter values (to be updated) are collected in $\boldsymbol{p} \in \mathbb{R}^{n_p}$, where n_p denotes the number of updating parameters. Note that, in this work, physically interpretable updating parameters directly available from the Equations of Motion (EoMs), e.g., spring and damping constants (e.g. representing mechanical connections), are used. Furthermore, although in this paper mechanical systems are considered, the presented methodology is, in fact, applicable to general nonlinear dynamic systems.

Values of updating parameters initially used in the reference model may be inaccurate and/or may be subject to change over time with respect to a measured physical system. Therefore, we define an unknown set of true parameter values, $\bar{\boldsymbol{p}} \in \mathbb{R}^{n_p}$, that, when used to parameterize the reference model in (1), exactly replicates the physical system. Here, it is assumed that the structure of the reference model is correct, i.e., functions $\boldsymbol{g}(.)$ and $\boldsymbol{h}(.)$ fully capture the dynamics of the physical system, and that discretization errors are negligible. Therefore, these true dynamics, in which unbiased measurement noise $\bar{\boldsymbol{w}}(t) \in \mathbb{R}^{n_{\text{out}}}$ is present, are given by

$$\begin{aligned} \dot{\bar{\boldsymbol{x}}}(t) &= \boldsymbol{g}(\bar{\boldsymbol{x}}(t), \boldsymbol{u}(t), \bar{\boldsymbol{p}}), \\ \bar{\boldsymbol{y}}(t) &= \boldsymbol{h}(\bar{\boldsymbol{x}}(t), \boldsymbol{u}(t), \bar{\boldsymbol{p}}) + \bar{\boldsymbol{w}}(t), \end{aligned} \tag{2}$$

where the bars indicate that variables are related to the true/physical system.

Since nonlinear models are evaluated, output features based on transient responses are required to compare the model and physical system. Therefore, an experiment (for proof of principle and/or a robust study of the method, also simulated experiments can be used) is performed with known excitation signals $\boldsymbol{u}(t)$ and initial conditions $\bar{\boldsymbol{x}}(t=0) = \boldsymbol{x}_0$. Measuring the output of the physical system in (2) yields sampled time series data $\bar{\mathtt{y}}_i$ for the i^{th} output signal $\bar{y}_i(t)$:

$$\bar{\mathtt{y}}_i = \left[\bar{y}_i(t_1), \ldots, \bar{y}_i(t_N))\right], \tag{3}$$

where N indicates the number of equidistant time samples and $i = 1, \ldots, n_{\text{out}}$. Simulating this experimental scenario using the reference model in (1), parameterized with an arbitrary set of parameter values $\boldsymbol{p}$, yields time series data $\mathtt{y}_i(\boldsymbol{p})$, defined similarly to (3). If $\boldsymbol{p} \neq \bar{\boldsymbol{p}}$, there exists a mismatch between $\mathtt{y}_i(\boldsymbol{p})$ and $\bar{\mathtt{y}}_i$, even if the measurement noise $\bar{w}$ in (2) is negligible. In the following, inverse mapping models will be used to reduce this mismatch by accurately estimating $\bar{\boldsymbol{p}}$ in a computationally fast manner.

To improve performance of the updating procedure, i.e., to make it computationally more efficient and/or to improve accuracy of the updating parameter estimates, a set of $n_{\psi,i} \leq N$ features $\bar{\boldsymbol{\psi}}_i \in \mathbb{R}^{n_{\psi,i}}$ is defined that captures important characteristics of the time series data:

$$\bar{\boldsymbol{\psi}}_i = \mathcal{L}(\bar{\mathtt{y}}_i) \quad \text{for } i = 1, \ldots, n_{\text{out}}, \tag{4}$$

where $\mathcal{L}$ represents a feature extraction function.

Using these features, the goal is to estimate a set of parameters $\hat{\boldsymbol{p}}$ (in real-time) that approximates $\bar{\boldsymbol{p}}$ by solving

$$\hat{\boldsymbol{p}} = \arg\min_{\boldsymbol{p}} \mathcal{E}\left(\bar{\boldsymbol{\psi}} - \boldsymbol{\psi}(\boldsymbol{p})\right), \tag{5}$$

where, with some abuse of notation, $\boldsymbol{\psi}_i(\boldsymbol{p}) := \boldsymbol{\psi}_i(\mathtt{y}_i(\boldsymbol{p}))$. Furthermore, $\mathcal{E} : \mathbb{R}^{n_\psi} \to \mathbb{R}$ is a cost functional and $\bar{\boldsymbol{\psi}}$ contains the measured features of all n_{out} output signals:

$$\bar{\boldsymbol{\psi}} = [\bar{\boldsymbol{\psi}}_1^\top, \ldots, \bar{\boldsymbol{\psi}}_{n_{\text{out}}}^\top]^\top, \tag{6}$$

such that $\bar{\boldsymbol{\psi}} \in \mathbb{R}^{n_\psi}$, with $n_\psi = \sum_{i=1}^{n_{\text{out}}} n_{\psi,i}$. Furthermore, note that $\boldsymbol{\psi}(\boldsymbol{p})$ is a set of features obtained equivalently using a simulation of the reference model (1) parameterized with the parameter set $\boldsymbol{p}$.

Since the objective is to solve (5) in (near) real-time, conventional, iterative methodologies that require multiple model evaluations or simulations usually are computationally too inefficient. Therefore, in the following section, an alternative method to update parameter values in (near) real-time using an IMM is presented that simultaneously meets the other requirements defined in Section 1, i.e., updating *interpretable* parameter values and updating *nonlinear* dynamical models.

2.2 Inverse mapping model

To solve the minimization problem in (5) with low computational effort in an online setting, inverse mapping models are employed, as proposed in [47]. In contrast to forward, first principles-based models, such as the reference model in (1), the input to an IMM is a set of (measured) output features and its output is the set of corresponding inferred (i.e., estimated) parameter values $\hat{\boldsymbol{p}}$. Mathematically, the IMM is represented by the function $\mathcal{I} : \mathbb{R}^{n_\psi} \to \mathbb{R}^{n_p}$:

$$\hat{\boldsymbol{p}} = \mathcal{I}(\bar{\boldsymbol{\psi}}), \tag{7}$$

where it is emphasized that, in general, $\mathcal{I}(\bar{\boldsymbol{\psi}}) \neq \bar{\boldsymbol{p}}$ due to the influence of noise and the approximate nature of the IMM. To constitute the function $\mathcal{I}$, a feed-forward artificial neural network (abbreviated as ANN and also known as the multilayer perceptron) is trained in an offline phase. These ANNs are renowned for their universal function approximation capabilities [54]. Moreover, in the online phase, inferring parameter values using ANNs is computationally efficient, as is also discussed in [51], where ANNs are used as surrogates for first-principles models.

Next, Section 2.2.1 gives a general introduction to feed-forward ANNs including a brief discussion of the accompanying mathematics. Afterwards, the offline and online phases of the IMPU method are discussed in Sections 2.2.2 and 2.2.3, respectively.

2.2.1 Feed-forward ANN

As indicated by (7), a (feed-forward) ANN maps a vector containing features $\bar{\boldsymbol{\psi}}$ (inputs to the ANN) to a vector containing parameter values $\hat{\boldsymbol{p}}$ (outputs of the ANN). To constitute this mapping, a (fully connected) feed-forward neural network uses multiple layers consisting of neurons that are all interconnected (between two neighboring layers) by trainable weights, see Figure 1. Each entry $\bar{\boldsymbol{\psi}}[j],\ j \in \{1, \ldots, n_\psi\}$ in the feature vector $\bar{\boldsymbol{\psi}}$ is assigned its own neuron (or node) in the input layer. Similarly, each entry $\hat{\boldsymbol{p}}[j],\ j \in \{1, \ldots, n_p\}$ of the parameter vector $\hat{\boldsymbol{p}}$ is assigned a neuron in the output layer. Between the input and output layer, there are n_r hidden layers, where layer $r \in \{1, \ldots, n_r\}$ has $n_z^{(r)}$ neurons. A large number of neurons enables the ANN to learn a more complex mapping. Although, theoretically, a single-hidden layer network is able to approximate any function (provided sufficient neurons) [55], using multiple layers with fewer nodes may, however, increase training and inferring efficiency of the feed-forward ANN [56].

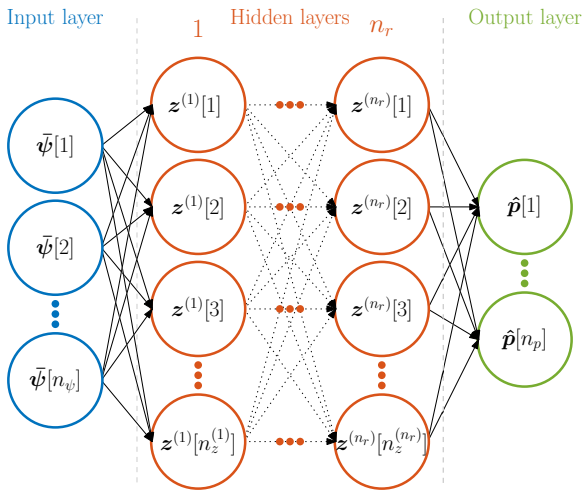

Fig. 1 Structure of a fully connected feed-forward ANN with n_r hidden layers.

The output value of the j^{th} neuron in (hidden) layer r is given by

$$\boldsymbol{z}^{(r)}[j] = \alpha(\boldsymbol{a}^{(r)}[j]), \tag{8}$$

where $\alpha(\cdot)$ represents an activation function that allows for nonlinearities to be introduced in the

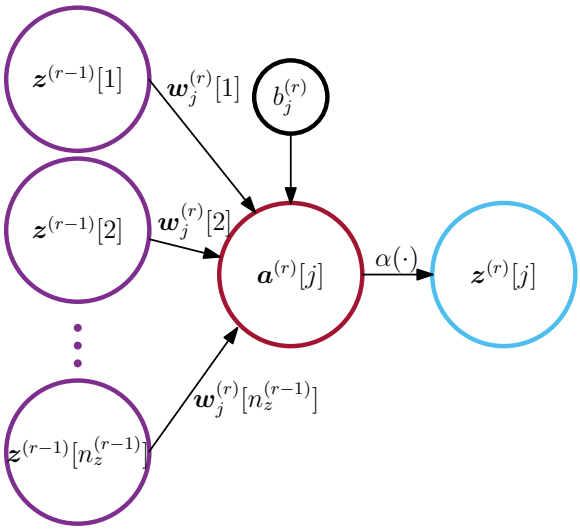

Fig. 2 Schematic representation of the calculation of the value of neuron j in layer r.

mapping and

$$\boldsymbol{a}^{(r)}[j] = \boldsymbol{w}_j^{(r)^\top} \boldsymbol{z}^{(r-1)} + b_j^{(r)}. \tag{9}$$

Here, $\boldsymbol{w}_j^{(r)} \in \mathbb{R}^{n_z^{(r-1)}}$ represents a vector with trainable weights between neuron j in layer r and all neurons in layer $r-1$. Note that layer $r=0$ denotes the input layer, i.e., $\boldsymbol{\psi}[j] =: \boldsymbol{z}^{(0)}[j]$ with $j \in \{1, \ldots, n_z^{(0)}\}$ and where $n_z^{(0)} = n_\psi$, and that layer $r = n_r + 1$ represents the output layer, i.e., $\boldsymbol{p}[j] =: \boldsymbol{z}^{(n_r+1)}[j]$ with $j \in \{1, \ldots, n_z^{(n_r+1)}\}$ and where $n_z^{(n_r+1)} = n_p$. Furthermore, the bias $b_j^{(r)}$ makes $\boldsymbol{a}^{(r)}[j]$ affine. In Figure 2, (8) and (9) are shown schematically.

2.2.2 Offline phase: Training of the ANN

In the offline phase, all weights in the ANN are trained such that the resulting ANN approximates the correct mapping and is consequently able to infer parameter values with high accuracy. To train the weights, supervised learning is employed for which n_s simulated pairs (or samples) of parameter values and their corresponding (output) features are generated. As shown schematically in Figure 3, depicting the offline phase of the IMPU method, to generate the training data, for each sample s, a set of updating parameter values $\boldsymbol{p}_s \in \mathbb{P} \subseteq \mathbb{R}^{n_p}$ is selected (based on, e.g., Latin HyperCube (LHC) sampling, see

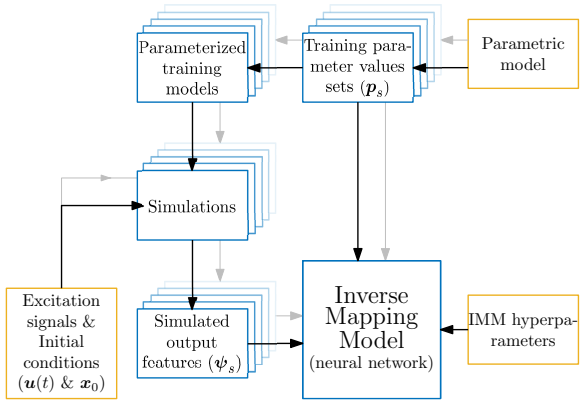

Fig. 3 Schematic representation of the offline phase of the IMPU method.

[57]). Here, $\mathbb{P}$ denotes a pre-defined admissible set of updating parameter values in which the true updating parameter values are expected to lie throughout the functional life of the physical system. Then, for each sample s, the forward reference model, i.e., (1), with $\boldsymbol{p} = \boldsymbol{p}_s$, is used to simulate an experiment, with predefined initial conditions $\boldsymbol{x}_0$ and excitations $\boldsymbol{u}(t)$, resulting in transient data $\mathrm{y}_i(\boldsymbol{p}_s)$, for $i = 1, \ldots, n_{\text{out}}$, from which a set of features $\boldsymbol{\psi}_i(\boldsymbol{p}_s)$ is extracted using (4). Note that the lack of hats and bars for $\boldsymbol{p}_s$ and $\boldsymbol{\psi}_i(\boldsymbol{p}_s)$ indicate that these variables correspond to training data. Combining the features from all n_{out} signals results in $\boldsymbol{\psi}(\boldsymbol{p}_s)$, see (6). Finally, to increase robustness of the ANN, the training data, i.e., $\boldsymbol{p}_s$ and $\boldsymbol{\psi}(\boldsymbol{p}_s)$ for $s \in \{1, \ldots, n_s\}$, are normalized such that, per entry in both $\boldsymbol{p}_s$ and $\boldsymbol{\psi}(\boldsymbol{p}_s)$, the minimum and maximum value across all n_s training samples equal 0 and 1, respectively. For more details about the normalization, see Appendix A.

Remark 1 To make the (training) data highly informative, the excitation signals $\boldsymbol{u}(t)$ employed to generate the output responses should have relevant frequency contents for the system under study, have a proper magnitude (to have a good signal-to-noise ratio), excite relevant nonlinearities, and take the bounds of the physical system into consideration, see, e.g., [29].

Having generated the normalized training data, the weights of the neural network are tuned iteratively by evaluating a randomly sampled subset of n_b training samples (called a batch). Here, the objective is to minimize the total mean

squared error between the training parameter values $\boldsymbol{p}_s$ and the parameter estimate of the current iterate k of the ANN, based on the corresponding training features, $\breve{\mathcal{I}}_k(\boldsymbol{\psi}(\boldsymbol{p}_s))$, for the batch of training samples. Here, the breve indicates that the inverse mapping function $\breve{\mathcal{I}}_k$ is still being trained and approaches the final inverse mapping $\mathcal{I}$ for increasing k until, for $k = n_k$, $\breve{\mathcal{I}}_k(\boldsymbol{\psi}(\boldsymbol{p}_s)) = \mathcal{I}(\boldsymbol{\psi}(\boldsymbol{p}_s))$. Here, n_k indicates the total number of weight tuning iterations applied during training.

This (total) mean squared error is also referred to as training loss and is defined as follows:

$$\epsilon_b = \frac{1}{n_p} \sum_{s \in \mathcal{B}} (\breve{\mathcal{I}}(\boldsymbol{\psi}(\boldsymbol{p}_s)) - \boldsymbol{p}_s)^\top (\breve{\mathcal{I}}(\boldsymbol{\psi}(\boldsymbol{p}_s)) - \boldsymbol{p}_s), \tag{10}$$

where $\mathcal{B}$ contains n_b unique randomly sampled training samples, such that $\mathcal{B} = \{s \in \mathbb{N} | 1 \leq s \leq n_s\}$. Note that, in contrast to (5), (10) minimizes the difference between the inferred and known training parameter values instead of features, which is caused by the inverse nature of the mapping. Minimizing ϵ_b in (10) is achieved using backpropagation, in which the derivative of ϵ_b with respect to all weights is used to find the optimal change in weights. After updating the weights, a new batch is sampled from the training samples (without reusing samples) and the backpropagation step is repeated until the first epoch is finished, i.e., until all training samples have been used once. Then, a new epoch is started and weights continue to be updated until a user user-specified number n_e of epochs have been completed. Note that the total number of weight tuning iterations n_k is thus defined by the number of executed epochs and the number of batches; $n_k = n_e \lceil \frac{n_s}{n_b} \rceil$. Training may be stopped earlier when detecting that the validation loss has not decreased for a user-specified number of successive epochs (referred to as patience-based early stopping [58]). Here, the validation loss is defined as the mean squared error for all samples in the validation data set. The validation data is obtained in an equivalent manner as the training data (with distinctively sampled parameter values) and used to evaluate performance of the ANN without being biased towards the training data. For more information about backpropagation and ANN training in general, the reader is referred to [54].

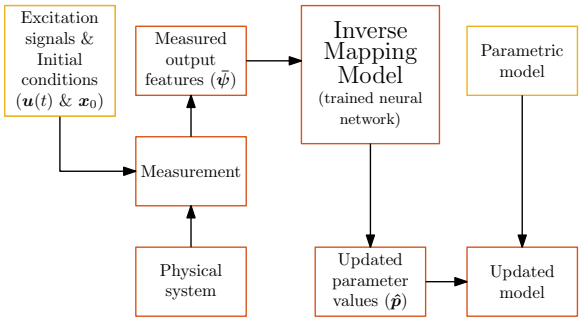

Fig. 4 Schematic representation of the online phase of the IMPU method.

2.2.3 Online phase

After having trained the ANN offline with simulated data, in the online phase, leading to the trained mapping in (7) this trained ANN is used to infer parameter values $\hat{\boldsymbol{p}}$ from measured data using (7), where the hat indicates that this is an inferred estimate. As shown in Figure 4, this is achieved by feeding output features $\bar{\boldsymbol{\psi}}$ to the ANN as obtained from a measurement of a physical system, see (7). Recall that, an overbar is used to indicate that these features relate to the physical system. The inferred (or updated) physical parameter values are then substituted in the parameterized reference model (1) resulting in the updated model. Note that, in the online phase, again normalized features and parameter values are utilized. To maintain consistency, the normalization in the online phase is performed using the same mapping as in the training phase, see Appendix A.

Due to the use of transient data to accommodate for nonlinear systems, in contrast to (modal) characteristics of linear systems as previously used in [47, 49, 50], the measured experiment in the online phase should be identical to the experiments simulated in the offline phase. Therefore, the initial conditions and excitation signals used in both phases are required to be equal.

Remark 2 In a practical setting, a homing procedure should be used to achieve the desired initial conditions. In applications where, during operation of the physical system, a certain action needs to be performed repetitively, the corresponding initial conditions and excitation signals can be used to constitute the aforementioned experiment, enabling on-the-fly parameter updating.

3 Definition of transient-based output features

In this section, different types of (not yet normalized) features $\boldsymbol{\psi}_i(\boldsymbol{p})$, extracted from sampled transient data $\mathtt{y}_i(\boldsymbol{p})$, are defined and discussed. In other words, different definitions for function $\mathcal{L}$ in (4) are introduced. Here, the aim is to define highly informative features, i.e., features that are sensitive to variations in output signals resulting from updating parameter variation. We aim to do so by exploiting engineering knowledge such that accuracy and computational efficiency of the IMPU method are improved. For the remainder of this section, we use the notation for the offline simulated (training) data (without overbar). However, the discussed feature extraction methodologies are applicable to the online measured data as well.

The introduced feature types are divided in time- and frequency-domain features, discussed in Sections 3.1 and 3.2, respectively.

3.1 Time-domain features

Here, feature types are introduced that directly use the sampled transient data $\mathtt{y}$.

Time samples

Time Samples (TS) features simply refer to the case in which a selection of n_{TS} entries of the sampled output signal are used as features:

$$\boldsymbol{\psi}_{\mathrm{TS}} = \mathtt{y}[\boldsymbol{v}_{\mathrm{TS}}], \tag{11}$$

with $\boldsymbol{v}_{\mathrm{TS}} \in \mathbb{N}^{n_{\mathrm{TS}}}$ denoting a vector containing the indices of the selected entries in $\mathtt{y}$. In this work, $\boldsymbol{v}_{\mathrm{TS}}$ is chosen such that the feature vector consists of time samples measured at equidistant moments in time. Note that, as the initial conditions are identical for all simulations and experiments, the measurements at initial time $t = 0$ are not included in the feature vector. Using TS features, the ANN is provided with the raw time series data as directly obtained from the measurements. Therefore, without exploiting engineering knowledge, the ANN has to implicitly extract 'hidden features of the underlying system' from the time series data in its hidden layer(s). In contrast, we will later introduce feature extraction methods that exploit engineering knowledge to aid the ANN by directly providing more informative data about the system. Specifically, we aspire to extract (smaller) sets of features that have higher information density, thereby improving accuracy and computational efficiency of the offline training of the ANN and the online inference of parameter estimates using the ANN.

Time extrema

For Time Extrema (TE) features, we only use the magnitude at and temporal location of a selection of n_{TE} local extrema in the time domain signal:

$$\boldsymbol{\psi}_{\mathrm{TE}} = \left[\mathtt{y}[\boldsymbol{v}_{\mathrm{TE}}]^\top, \mathtt{t}[\boldsymbol{v}_{\mathrm{TE}}]^\top\right]^\top. \tag{12}$$

Here, $\mathtt{t} = [t_1, \ldots, t_N]$ consists of all time instances at which the measured signal is sampled, and $\boldsymbol{v}_{\mathrm{TE}} \in \mathbb{N}^{n_{\mathrm{TE}}}$ contains the (ascending) indices of the first n_{TE} extrema that are identified using some extrema-finding algorithm. Note here that local extrema, i.e., local minima and local maxima, are defined as the local maxima/minima near the peaks and/or troughs of the signals, such that, despite the influence of noise, only one extremum is found per peak/trough. In practice, this can be achieved by using various options of the findpeaks algorithm in Matlab such as the minimum peak prominence (a user-specified minimum change in signal value required on either side of the peak before the signal attains a higher value than at the peak itself [59]). As, for different samples obtained for distinct parameter values sets, the measured signals may not have the same number of extrema, n_{TE} should be equal to or smaller than the lowest number of extrema encountered in the output signal across all training samples.

An advantage of TE features is that, especially if the transient signal is similar to a free response, the height and temporal location of extrema in the transient signal are related to dynamic characteristics such as amplitude decay of eigenmodes and eigenfrequencies, respectively. Consequently, these are informative about physical parameters such as mass, damping, and stiffness parameters. A disadvantage, however, is that the temporal location of an extremum is sensitive to noise, potentially causing the true location of the extremum to be identified at one of the neighbouring entries in $\mathtt{t}$. Due to the discrete nature of

$\mathtt{t}$, this may cause relatively large variations in the resulting feature values.

3.2 Frequency-domain features

In this section, features are extracted from frequency domain data $\mathtt{Y} \in \mathbb{C}^{N/2+1}$, where N is an even integer such that we obtain $N/2-1$ complex numbers and 2 real numbers (for $f = 0$ and half the Nyquist frequency). For this, first, the time domain data $\mathtt{y}$ is transformed to the frequency domain:

$$\mathtt{Y}(f_j) = \mathcal{F}(\mathtt{y}(t_k)), \tag{13}$$

where t_k denotes timestep k (where $k = 1, \ldots, N$), f_j denotes frequency bin j (where $j = 1, \ldots, N/2+1$), and $\mathcal{F}$ represents the Discrete Fourier Transform (DFT); in practice, usually the Fast Fourier Transform (FFT) is used.

Since the discrete frequency domain data is represented in a set of complex numbers, there are two options to pass this data to the neural network (which requires real valued data): 1) using the Magnitude and Phase information (MP): see (14), or 2) using the Real and Imaginary components of the frequency domain data (RI): see (15):

$$\boldsymbol{\psi}_{\mathrm{MP}} = \left[|\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]|^\top, \angle \mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]^\top \right]^\top, \tag{14}$$

$$\boldsymbol{\psi}_{\mathrm{RI}} = \left[\Re\left(\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]\right)^\top, \Im\left(\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]\right)^\top \right]^\top, \tag{15}$$

where $|\cdot|$, $\angle(\cdot)$, $\Re(\cdot)$, and $\Im(\cdot)$, represent the magnitude, phase, real part, and imaginary part of a complex number, respectively. Since the phase and imaginary part at $f_1 = 0$ and at the folding frequency (half of the sample frequency) of real valued time signals are always zero, these numbers do not convey any information. Consequently, they are not included in the feature vector.

Although the discrete frequency domain signal contains, by definition, exactly the same information as the discrete time domain signal, the relevant information may be more condensed in the frequency domain signal. For example, frequencies at which the measured signal has only small magnitudes, are likely less informative about the system's dynamics and response than frequencies with high magnitudes. Indices of the frequency bins that are deemed informative, e.g., located near eigenfrequencies of the linearized system or as encountered in exploratory experiments, are therefore collected in the index set $\boldsymbol{v}_{\mathrm{FT}} \in \mathbb{N}^{n_{\mathrm{FT}}}$. Only using the features belonging to these more relevant frequencies bins, referred to as Frequency Bins of Interest (FBoIs), leads to a more compact set of features, raising potential for improved performance of the ANN. In this paper, FT-based features that are limited to the information in some FBoIs are indicated by either MP (FBoI), or RI (FBoI).

4 Illustrative case study

In this section, the inverse mapping parameter updating method and various feature types are applied on a case study of a nonlinear multi-body system. Here, we employ simulated experiments, where noise will be added to the response signals. Firstly, the system and experiment design are introduced, along with exploratory simulations, in Section 4.1. Additionally, this section discusses the employed ANN and accompanying (training) settings. In Section 4.2, results are analyzed in terms of accuracy of the inferred parameter values and responses of the updated model. Finally, an extensive analysis of the generalization capabilities of the ANN is given in Section 4.3.

4.1 Problem setting

4.1.1 Model description

Figure 5 shows the side view of the multi-body system academic demonstrator by means of which the introduced methodology is demonstrated. The system consists of one translating, rigid beam (beam 1) with mass m_1, which is connected to a rotating, rigid beam (beam 2) with mass m_2 and mass moment of inertia $I_{\mathrm{CoM},2}$ about its Center of Mass (CoM). The position of beam 1 is indicated by $y_1(t)$ [m] and the orientation of beam 2 is indicated by $y_2(t)$ [rad]. Here, the location of CoM of beam 2 is given by the position vector pointing from the joint connecting beams 1 and 2 to the CoM of beam 2: $\boldsymbol{\rho}_{\mathrm{CoM}} = (l_{\mathrm{CoM},1}\cos(y_2) - l_{\mathrm{CoM},2}\sin(y_2))\,\boldsymbol{e}_1 + (l_{\mathrm{CoM},1}\sin(y_2) + l_{\mathrm{CoM},2}\cos(y_2))\,\boldsymbol{e}_2$, where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ represent unit vectors of a (global) fixed reference frame as indicated in Figure 5.

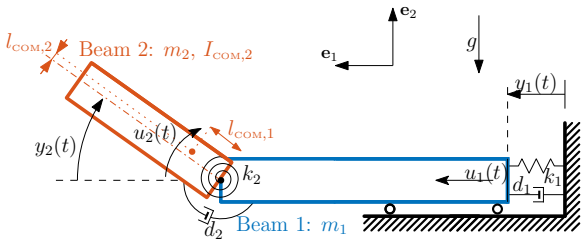

Fig. 5 Schematic representation of the rigid double beam academic demonstrator model.

Table 1 Updating parameter space bounds.

Parameter	Lower bound	Upper bound	Unit
d_1	0.8	1.2	$\frac{\text{N·s}}{\text{m}}$
d_2	1.75×10^{-4}	2.25×10^{-4}	$\frac{\text{N·m·s}}{\text{rad}}$
k_1	5	10	$\frac{\text{N}}{\text{m}}$
k_2	2.7×10^{-2}	4.5×10^{-2}	$\frac{\text{N·m}}{\text{rad}}$

The translational and rotational joint have stiffness and damping constants k_1 and d_1, and k_2 and d_2, respectively. The rest length/rotation of springs k_1 and k_2 are l_{k_1} and θ_{k_2}, respectively. Furthermore, the system is subject to gravity, with gravitational acceleration constant g. For reference, the EoMs for this system are presented in Appendix B.

In the following case study, the values of parameters d_1, d_2, k_1, and k_2 are assumed to be uncertain and, therefore, will be updated, i.e., $\boldsymbol{p} = [d_1, d_2, k_1, k_2]^\top$. Here, we assume that the reference model is parameterized with an initial guess for these updating parameters $\boldsymbol{p}_c$. Additionally, for this case study, it is assumed that the true parameter values may vary, with equal amounts in both the negative and positive direction, around this initial guess. Consequently, the upper and lower bounds of the updating parameter space $\mathbb{P}$, tabulated in Table 1, are chosen such that $\boldsymbol{p}_c$ lies in the center of $\mathbb{P}$. The values for all other parameters remain constant and are given in Table 2.

Beams 1 and 2 are excited, in open-loop, by force $u_1(t)$ and torque $u_2(t)$, respectively:

$$u_1(t) = \begin{cases} 5 \text{ N} & \text{if } 0.2 \leq t < 0.25, \\ 0 \text{ N} & \text{else,} \end{cases}$$
$$u_2(t) = \begin{cases} 0.075 \text{ Nm} & \text{if } 0.2 \leq t < 0.25, \\ 0 \text{ Nm} & \text{else.} \end{cases} \tag{16}$$

Table 2 Constant parameter values.

Parameter	Value	Unit
m_1	2.1630	kg
m_2	0.13701	kg
$I_{\text{CoM},2}$	2.981×10^{-4}	kg·m^2
$l_{\text{CoM},1}$	3×10^{-3}	m
$l_{\text{CoM},2}$	4.6×10^{-3}	m
l_{k_1}	0.05	m
θ_{k_2}	0	rad
g	9.81	$\frac{\text{m}}{\text{s}^2}$

Here, impulse-like excitations are used since these yield free vibration responses that vary significantly when parameters of the system are changed. Additionally, in the time interval $[0, 0.2]$ s, no excitations are imposed such that information with respect to the static equilibrium of the system is also obtained. In future work, the excitation signals may be optimized to increase the sensitivity of the response with respect to parameter changes, see e.g., [60, 61].

Furthermore, as initial conditions, the static equilibrium of the system is used, which is parameterized with updating parameter values in the center of the parameter space $\mathbb{P}$ (referred to as $\boldsymbol{p}_c$), i.e., $\boldsymbol{y}(t=0) = \begin{bmatrix} 0.05 & -0.1898 \end{bmatrix}^\top$ and $\dot{\boldsymbol{y}}(t=0) = \begin{bmatrix} 0 & 0 \end{bmatrix}^\top$.

Using numerical integration[1], the system is simulated for a time interval of 2 s, in which $N = 256$ equidistant samples of $y_1(t_k)$, and $y_2(t_k)$, $k \in \{1, \ldots, N\}$, are obtained as outputs. Here, N is chosen as a power of 2 to make the Fast Fourier Transform (FFT) algorithm as efficient as possible. To mimic the (simulated) measurements, zero-mean Gaussian noise is added to these outputs with standard deviations $\sigma_{y_1} = 2 \times 10^{-4}$ m (approximately 0.5% of maximum magnitude of y_1) and $\sigma_{y_2} = 2 \times 10^{-2}$ rad (approximately 2% of maximum magnitude of y_2). In practice, these noise properties should be chosen corresponding to the (expected) noise observed in measurements performed on the physical system. Note that, as explained in [62], the addition of noise to training data improves robustness of the ANN. Examples of possible responses using the above experiment design are provided in Section 4.1.2.

[1]For the numerical integration, the ode45 function (with `RelTol`=1×10^{-3}, `AbsTol`=1×10^{-6}) in Matlab 2021b has been used.

To provide the reader with some insight into the dynamic characteristics of the system, properties of the linearized system for five different combinations of updating parameter values, also referred to as parameter cases and tabulated in Table 3, are presented. Note that parameter case 1 refers to $\boldsymbol{p}_c$ and the values of cases 2, 3, 4, and 5 are located at the boundaries of $\mathbb{P}$, as defined in Table 1, such that the 'most extreme' possible behavior of the system is presented here. Each model, parameterized using one of the parameter cases, is linearized around its static equilibrium point. Resulting eigenvalues (with positive imaginary part) λ_i, (damped) eigenfrequencies $f_{i,\text{eig}}$, and their corresponding, modal, damping coefficients ξ_i, with $i = 1, 2$ indicating the first and second eigenmodes, are listed in Table 3. More information about the linearized systems, e.g., eigenmodes and Frequency Response Functions (FRFs), is provided in Appendix C.

4.1.2 Output features for exploratory simulations

In this section, exploratory simulation results are shown for the five updating parameter cases, as defined in Table 3. These simulations are performed according to the simulated experiment design (excitation signals and initial conditions) introduced in Section 4.1.1. Furthermore, features are extracted, conform the theory in Section 3, and plotted for additional insight.

Some settings to extract the features are as follows. For the time samples features, two variants are used: 1) dense with $n_\psi = N = 256$, and 2) sparse with $n_\psi = 34$. For the time extrema features, the minimum peak prominence $\texttt{mpp}(\texttt{y})$ is used for output signal $\texttt{y}$ (as used in the timepeaks Matlab function [59]) and is defined as

$$\texttt{mpp}(\texttt{y}) = 0.15\left(\max(\texttt{y}) - \min(\texttt{y})\right), \tag{17}$$

where $\texttt{y}$ denotes a sampled output signal. As a result, a peak is only defined if the value of $\texttt{y}$ between that peak and the closest peak with a higher magnitude drops by a minimal amount of $\texttt{mpp}$ [59]. Note that the definition of $\texttt{mpp}(\texttt{y})$ should be tailored towards the encountered responses using engineering insight. Due to the different dominating frequencies of both output signals, the number of peaks in output signals $y_1(t)$ and $y_2(t)$ are $n_{\text{TE},1} = 1$ and $n_{\text{TE},2} = 5$, respectively. Furthermore, the FBoIs are defined as the frequency bins within the frequency interval from 0 Hz to 4 Hz, such that the (damped) eigenfrequencies of the linearized models, as given in Table 3, lie within this range.

In Figures 6 and 7, time-domain responses, here referred to as TS (dense) features, are shown for the various parameter cases. In addition, TS (sparse) and TE features are indicated by color-coded squares in Figures 6 and 7, respectively. Furthermore, Figures 8 and 9 show the corresponding Fourier transforms of the TS (dense) response features using the magnitude and phase representation, and the real and imaginary values representation, respectively. Additionally, Figures 8 and 9 indicate the features corresponding to the FBoIs using color-coded squares.

4.1.3 Artificial neural network

As mentioned in Section 2.2, the IMM is constituted by a (trained) feed-forward ANN, see Figure 1. In this case study, a 4-layer fully connected feed-forward ANN, i.e., $n_r = 2$, is used of which the number of neurons and activation function per layer are listed in Table 4. Here, the Rectified Linear Unit activation function, defined as

$$\alpha(a) = \begin{cases} a & \text{if } a > 0, \\ 0 & \text{else,} \end{cases} \tag{18}$$

is abbreviated as ReLU. In the output layer, linear activation functions ($\alpha(a) = a$) are used to enable inferring of parameters outside the training parameter space $\mathbb{P}$, which is, if the bounds of $\mathbb{P}$ are used for normalization, impossible using, e.g., Sigmoid activation functions. Note that a single multi-output ANN is used as this is typically more efficient than having a single-output ANN per updating parameter [63].

The ANN is trained using a collection of $n_s = 1000$ training samples, of which the corresponding updating parameter values are sampled from $\mathbb{P}$ using a Latin HyperCube. Optimization of the weights and biases of all neurons is done using the Keras Adams algorithm [64], where the mean squared error is minimized over 200 epochs in 20 batches of 50 samples. Additionally, patience-based early stopping is included, meaning that training is stopped if the validation loss,

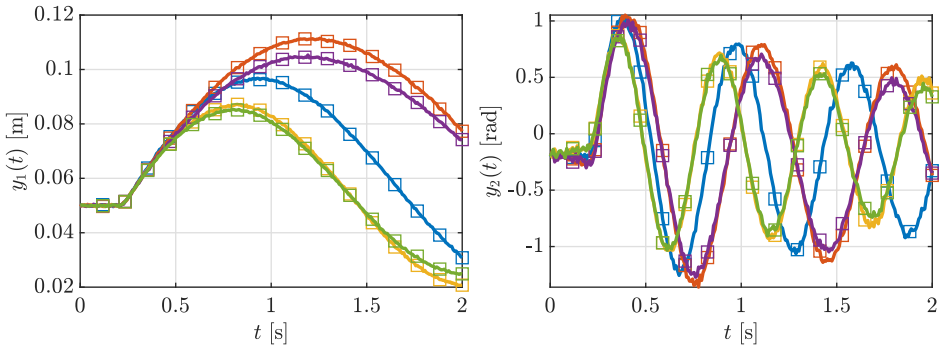

Fig. 6 Time domain simulation responses with temporally equidistant TS (dense) features (consisting of all points used to draw each response). Additionally, the TS (sparse) features are indicated by the (colored) squares (□). The (line) colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—).

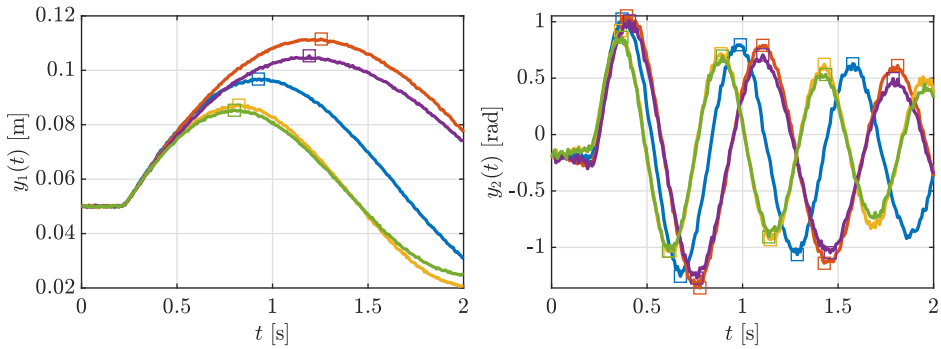

Fig. 7 Time domain simulation responses with TE features (local extrema) indicated by the (colored) squares (□). Note that the TE feature vector contains pairs of the temporal location and height of each extremum. The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—).

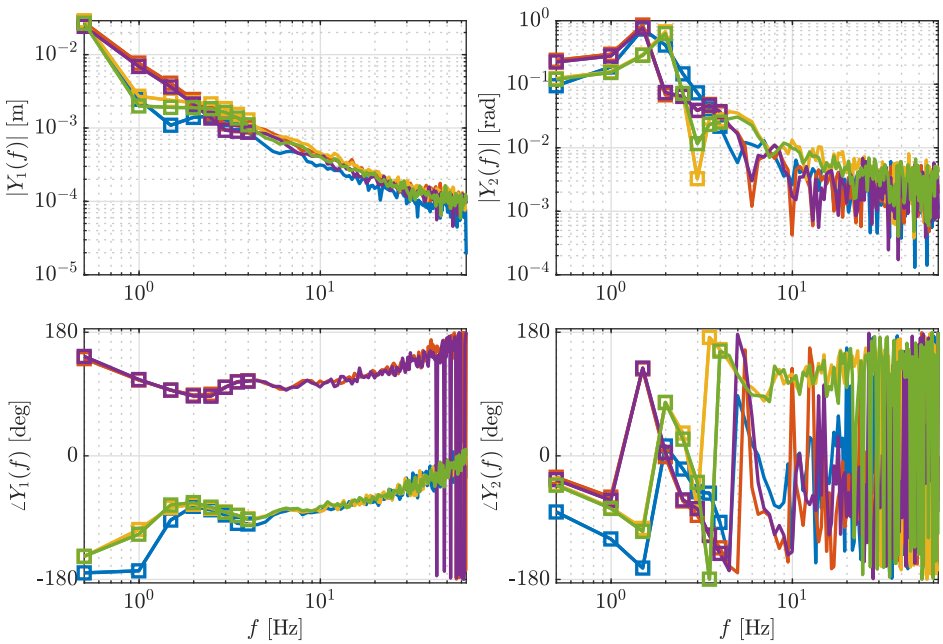

Fig. 8 Fourier transform of TS (dense) simulation responses represented by magnitude and phase. The MP features consist of all points used to draw each curve. Furthermore, the MP (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—).

Table 3 Different parameter cases with corresponding updating parameter values, resulting eigenvalues, (damped) eigenfrequencies, and modal damping coefficients of the linearized models. Note that, $\xi_i = \frac{-\Re(\lambda_i)}{2\pi f_{i,\text{eig}}}$ in principle is defined for proportionally damped systems, whereas the current (linearized) system is generally viscously damped. Nonetheless, here, it is used to provide insight in the relative amount of damping.

Parameter case	d_1 $[\frac{\text{N·s}}{\text{m}}]$	d_2 $[\frac{\text{N·m·s}}{\text{rad}}]$	k_1 $[\frac{\text{N}}{\text{m}}]$	k_2 $[\frac{\text{N·m}}{\text{rad}}]$	λ_1 [rad/s]	λ_2 [rad/s]	$f_{1,\text{eig}}$ [Hz]	$f_{2,\text{eig}}$ [Hz]	ξ_1 [-]	ξ_2 [-]
1	1.0	2.00×10^{-4}	10	3.6×10^{-2}	$-0.217 + 2.074j$	$-0.331 + 10.477j$	0.330	1.668	0.104	0.032
2	0.8	1.75×10^{-4}	5	2.7×10^{-2}	$-0.174 + 1.464j$	$-0.290 + 9.034j$	0.233	1.438	0.118	0.032
3	0.8	1.75×10^{-4}	15	4.5×10^{-2}	$-0.174 + 2.548j$	$-0.290 + 11.776j$	0.406	1.874	0.068	0.025
4	1.2	2.25×10^{-4}	5	2.7×10^{-2}	$-0.261 + 1.451j$	$-0.372 + 9.031i$	0.231	1.437	0.177	0.041
5	1.2	2.25×10^{-4}	15	4.5×10^{-2}	$-0.261 + 2.540j$	$-0.372 + 11.774j$	0.404	1.874	0.102	0.032

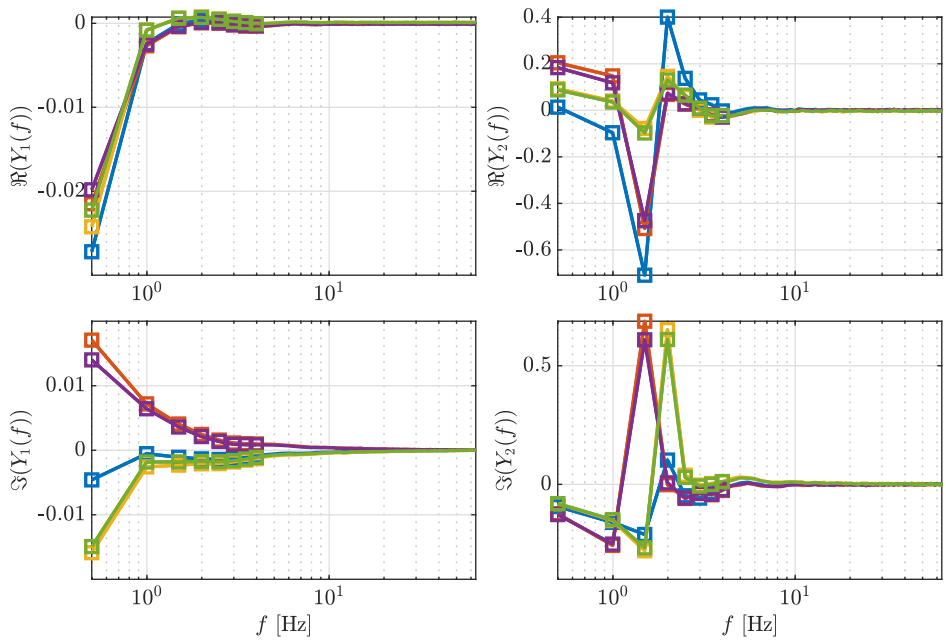

Fig. 9 Fourier transform of TS (dense) simulation responses represented using real and imaginary values. The RI features consist of all points used to draw each curve. Furthermore, the RI (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—).

calculated for 1000 validation samples, has not decreased in 40 successive epochs. Here, the validation samples are randomly selected from $\mathbb{P}$ using a uniform probability distribution. Note that the ANN hyperparameters, i.e., the ANN structure and training settings, used here have been chosen empirically. Although ANN hyperparameter tuning is not trivial, after some trial and error it has been found that the currently applied ANN structure and training settings perform relatively well. This being said, these hyperparameters are not expected to be optimal. Therefore, optimization of the ANN hyperparameters is still an open challenge.

Using a single core 2.60GHz CPU, the offline generation of the training data, i.e., simulating the reference model for all $n_s = 1000$ training samples, takes approximately 3 minutes. Additionally, the offline training of the ANN takes 13 s. In the online phase, inferring a single set of parameter values $\hat{\boldsymbol{p}}$ from a feature vector $\bar{\boldsymbol{\psi}}$ takes only 3 ms, indeed enabling (near) real-time parameter updating.

Table 4 ANN structure.

Layer	Number of neurons	Activation function
Input	n_{ψ}	-
Hidden 1	200	ReLU
Hidden 2	100	ReLU
Output	$n_p (= 4)$	Linear

4.2 Inference results: parameter updating using the ANN

In the following, the accuracy of the inferred parameter values and updated model are analyzed in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 Inferred parameter accuracy

To analyze the accuracy of the inferred parameter values, $\hat{\boldsymbol{p}}$, the reference model is simulated for $n_{\bar{s}} = 1000$ test samples. Note that, since, for testing, measurements are mimicked, we employ the overbar notation to indicate (the number of) test samples. To ensure an unbiased evaluation of the ANN performance, the test samples should be distinct from the training and validation samples. Therefore, for each test sample $\bar{s}$, the model is parameterized with a set of true parameters $\bar{\boldsymbol{p}}$ that are randomly selected in $\mathbb{P}$ again based on a uniform probability distribution. To mimic actual measurements, noise is added to the simulated outputs with equal properties as used to generate the training data. The features extracted from these simulated measurements are denoted as $\bar{\boldsymbol{\psi}}_{\bar{s}}$ and fed into the ANN, see (7), to obtain $\hat{\boldsymbol{p}}_{\bar{s}}$.

Accuracy of the inferred parameter values of test sample $\bar{s}$ is evaluated based on the relative error, $\boldsymbol{\varepsilon}_{\bar{s}} \in \mathbb{R}^{n_p}$, of the parameter estimate $\hat{\boldsymbol{p}}_{\bar{s}}$:

$$\boldsymbol{\varepsilon}_{\bar{s}} = (\hat{\boldsymbol{p}}_{\bar{s}} - \bar{\boldsymbol{p}}_{\bar{s}}) \oslash (\bar{\boldsymbol{p}}_{\bar{s}}), \tag{19}$$

where $\oslash$ denotes the element-wise division operator.

Before comparing results for different output features, we first focus on the case in which we only use the RI (FBoI) features as inputs to the ANN. Figure 10 displays the distribution of these relative errors (in %) per updating parameter in a histogram. Indeed, the ANN is able to infer parameter values from the provided features. The accuracy of the estimates varies per updating parameter, with d_1 being estimated with the lowest accuracy and k_2 with the highest accuracy. It is remarked that, despite a relatively larger contribution of noise in y_2 compared to y_1 (as mentioned in Section 4.1.1), the parameters most closely related to this signal, i.e., k_2 and d_2, are estimated with slightly higher accuracy than k_1 and d_1. Here, please note that, as is explained in Appendix C, the responses of the system are largely uncoupled. The observation that k_2 and d_2 are estimated more accurately, is most likely a result of the relatively low amount of oscillations in signal y_1 compared to y_2, as seen in Figures 6 and 9. Nevertheless, even for d_1, the worst encountered relative error is only approximately 10%, whereas the majority lies below 5%. Therefore, it can be concluded that the parameter values are inferred with good accuracy.

Additionally, in Figure 11, the (absolute) relative error for each test sample is plotted (in %) in the subspace of $\mathbb{P}$ spanned by d_1 and d_2. Note that the values of k_1 and k_2 vary randomly, within $\mathbb{P}$, for each plotted sample. This figure shows that accuracy at or near the edges of the d_1-d_2 subspace is slightly worse than in the middle. Specifically, for d_1 and d_2, there are relatively many high relative errors at the left and bottom edge, respectively. This is explained by the fact that, in these regions, there are obviously less training samples (actually, none beyond the bounds of the training space) that allow the ANN to properly learn the mapping between $\boldsymbol{\psi}$ and $\boldsymbol{p}$ in this part of the parameter space. To alleviate this phenomenon, the training space may be extended beyond the parameter values that are expected to be encountered on the physical system.

Having analyzed the results for the RI (FBoI) features only, in the following, different output feature types are compared. For ease of comparison, the performance of the IMPU method (for any feature type) is summarized using three relative error metrics, being the bias of the relative error, $\boldsymbol{\mu}_{\varepsilon}$, the (unbiased sample [65]) standard deviation, $\boldsymbol{\sigma}_{\varepsilon}$, and the average absolute relative error, $\boldsymbol{\mu}_{|\varepsilon|}$:

$$\boldsymbol{\mu}_{\varepsilon} = \frac{1}{n_{\bar{s}}} \sum_{\bar{s}=1}^{n_{\bar{s}}} \boldsymbol{\varepsilon}_{\bar{s}}, \tag{20}$$

$$\boldsymbol{\sigma}_{\varepsilon} = \sqrt{\frac{1}{n_{\bar{s}} - 1} \sum_{\bar{s}=1}^{n_{\bar{s}}} (\boldsymbol{\varepsilon}_{\bar{s}} - \boldsymbol{\mu}_{\varepsilon}) \otimes (\boldsymbol{\varepsilon}_{\bar{s}} - \boldsymbol{\mu}_{\varepsilon})}, \tag{21}$$

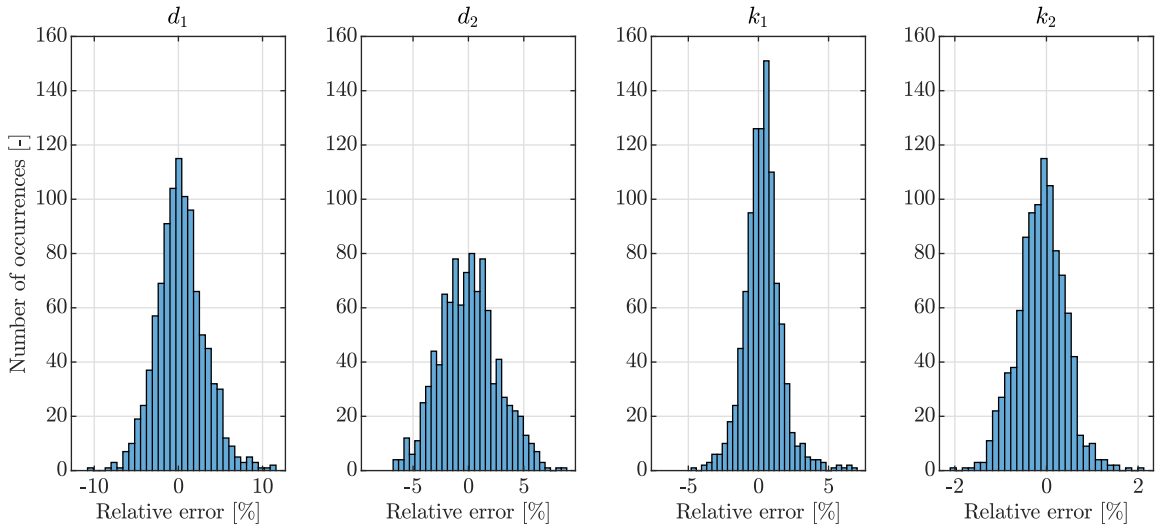

Fig. 10 Distribution, per updating parameter, of relative inferred parameter errors, $\boldsymbol{\varepsilon}_{\bar{s}}$, as obtained using the RI (FBoI) output features and $n_s = 1000$ training samples.

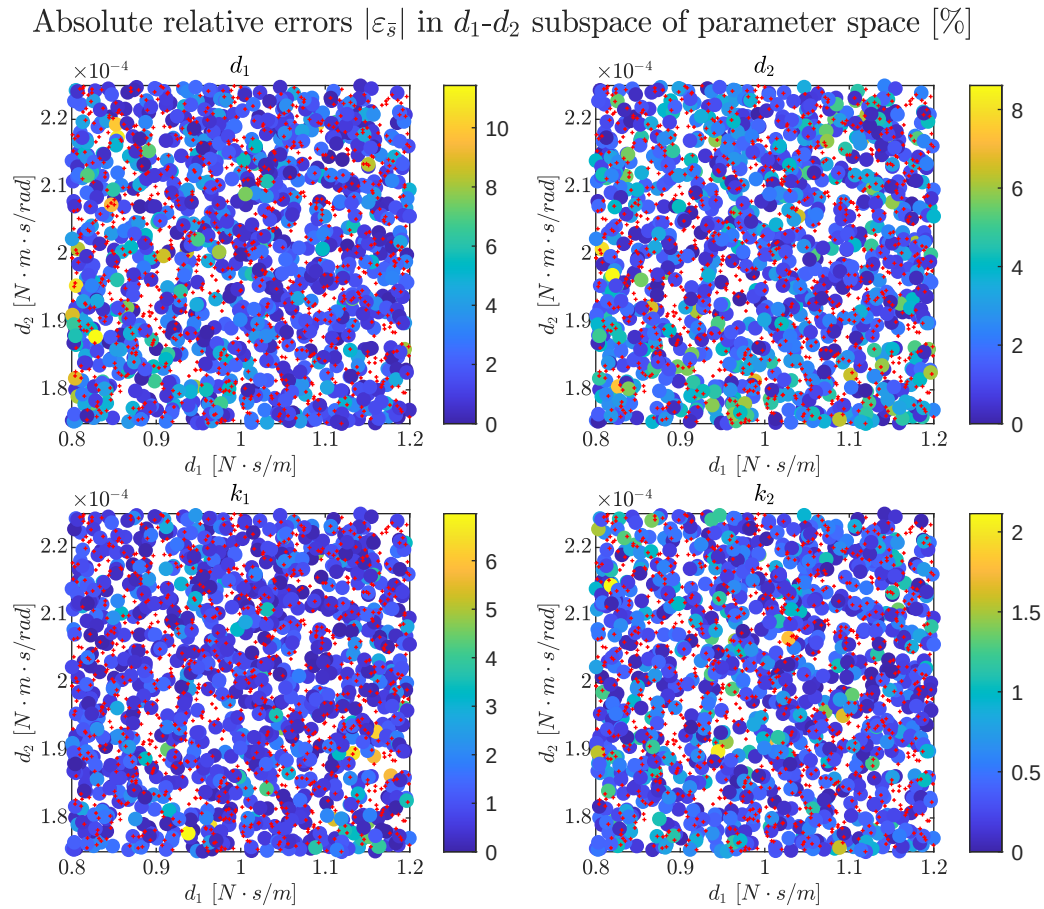

Fig. 11 Absolute relative error $|\varepsilon_{\bar{s}}|$ of each test sample $\bar{s}$ in the d_1-d_2 subspace of $\mathbb{P}$ per updating parameter, as obtained by using the RI (FBoI) output features with $n_s = 1000$. Red plus signs (+) indicate locations of training data samples.

$$\boldsymbol{\mu}_{|\varepsilon|} = \frac{1}{n_{\bar{s}}} \sum_{\bar{s}=1}^{n_{\bar{s}}} |\boldsymbol{\varepsilon}_{\bar{s}}|, \tag{22}$$

where $\otimes$ denotes the element-wise multiplication operator.

In Table 5, these metrics are listed (in %) per parameter for the output feature types introduced in Section 3. Firstly, it is observed that, in terms of $\boldsymbol{\sigma}_{\varepsilon}$ and $\boldsymbol{\mu}_{|\varepsilon|}$, the dense and sparse time samples feature vectors perform roughly equally well for the stiffness parameters. However, the sparse variant performs worse for the damping parameters. This might be attributed to the fact that the dense TS features are more likely to contain the extremum values of the transient signal which provide valuable information about damping, see Figure 6. Secondly, the results obtained using time extrema features are comparatively inaccurate, which can be ascribed to a relatively large influence of noise and low number of features. This holds especially for d_1 and k_1, as there is only one relevant extremum for these parameters due to the low eigenfrequency of the dominant mode for y_1, see Figure 7. Thirdly, for the frequency-domain based features, omitting the (uninformative and noisy) features outside the FBoIs does significantly improve performance with respect to the frequency-domain based features where all frequency bins are used, as was hypothesized in Section 3.2. Fourthly, MP features are less accurate than RI features, both when using features in all frequency bins, or those restricted to the FBoIs. As this difference holds primarily for d_1 and k_1, to explain this observation, we focus our attention on the first Fourier-transformed output signal Y_1 (recall the quite uncoupled nature of this particular system, see Appendix C). This difference in performance may be explained by the fact that some of the (normalized) MP features do not deviate significantly for distinct parameter values combinations. To illustrate this, in Figures 12 and 13, the MP (FBoI) and RI (FBoI) features are normalized between 0 and 1 such that the maximum and minimum feature values across the five parameter cases defined in Table 3 equal one and zero, respectively. In Figure 12, it is observed that the normalized features originating from $\angle Y_1$, i.e., part of the (normalized) MP features, are relatively similar for parameter combinations 2 and 4, and combinations 3 and 5. Therefore, the ANN has difficulty in distinguishing between each of these similar parameter cases, e.g., the normalized features for parameter combinations 2 and 4 are both approximately 1. In contrast, this is only the case to a lesser extent for normalized features originating from $\Im(Y_1)$ in Figure 13. Consequently, distinguishing between these parameter cases when using MP features is mostly done based on the $|Y_1|$ features, whereas for the RI features, both $\Re(Y_1)$ and $\Im(Y_1)$ can distinguish between these cases. Finally, comparing the TS features with the RI (FBoI) features, we see that the RI (FBoI) features yield better performance than the sparse set of TS features (with an identical number of features). Alternatively, the dense TS features yield comparable performance at the cost of 15 times as many features. Note, however, that still the RI (FBoI) features perform, although slightly, better. This, again, leads to the conclusion that, the individual RI (FBoI) features have relatively high informativeness.

It should be remarked that the conclusions drawn here are found for a specific system and (simulated) experiment design. Different systems or designs, e.g., systems with higher modal density, (highly) coupled DoFs, and insufficient actuation will influence accuracy of the individual features types. For example, time extrema features are expected to be less profitable to estimate damping properties related to higher-order (or less dominant) modes. Therefore, automated feature selection techniques have high potential to select the most informative features for any system and experiment design [66].

Nevertheless, some general conclusions are drawn. Firstly, by using transient-based output response features as input to an IMM, physically interpretable parameters of nonlinear dynamical models can be updated online with little computational effort while achieving acceptable levels of accuracy. Furthermore, it is shown that, at least for mechanical systems, defining features using engineering knowledge can increase performance of the ANN. For example, only using frequency-domain based features in the frequency ranges that are relevant for the dynamics of the system can result in higher accuracy of the inferred updating parameter values.

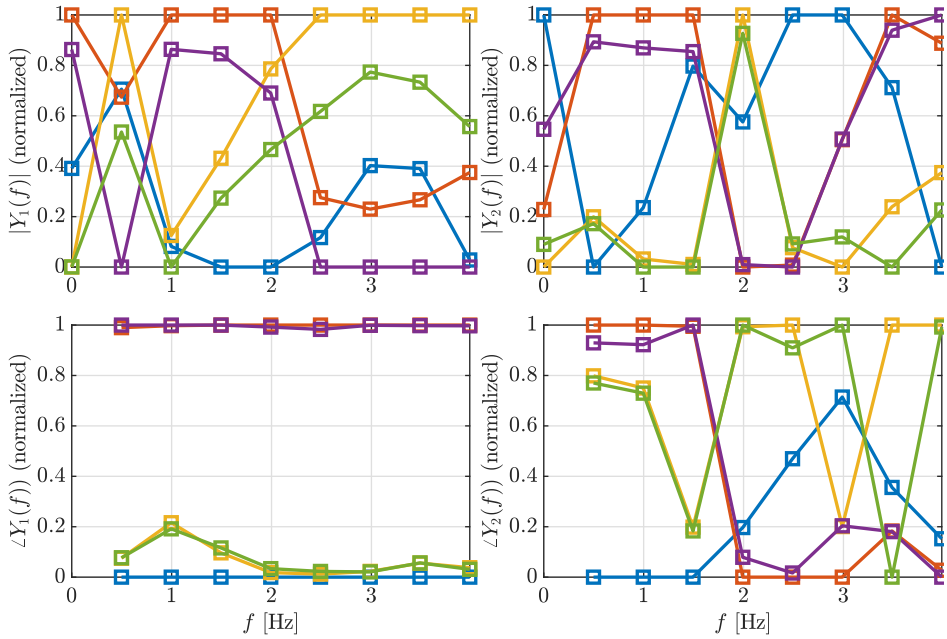

Fig. 12 MP (FBoI) features, normalized for the five parameter cases, as obtained from the Fourier transform of TS (dense) simulation responses represented using magnitude and phase, zoomed in on the FBoIs. The MP (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—).

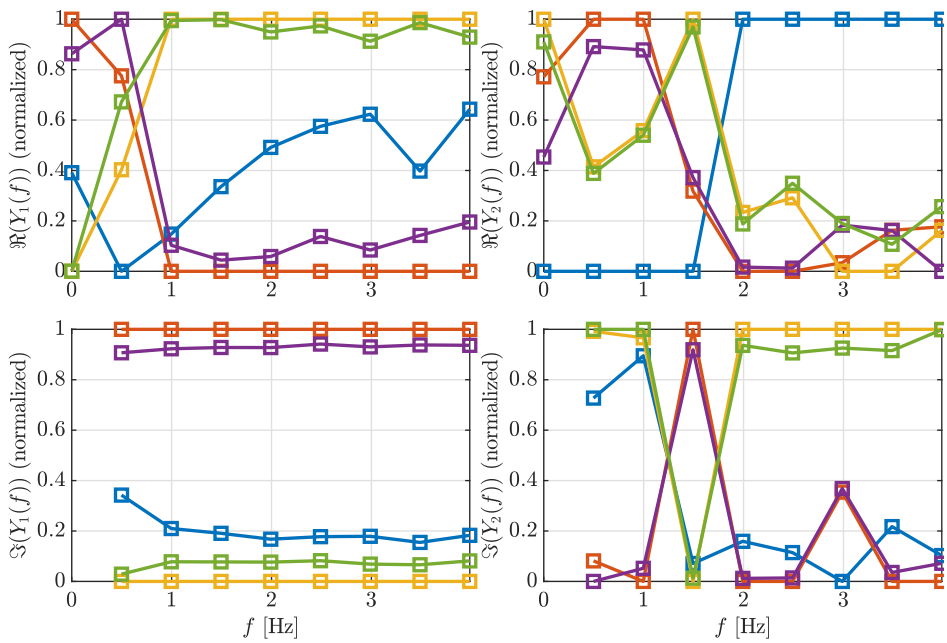

Fig. 13 RI (FBoI) features, normalized for the five parameter cases, as obtained from the Fourier transform of TS (dense) simulation responses represented using real and imaginary values, zoomed in on the FBoI. The RI (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—).

Table 5 Relative error metrics per updating parameter for all feature types, for $n_s = 1000$. Furthermore, the number of features per output feature type, n_ψ, is listed as well.

Feature Type	n_ψ	$\boldsymbol{\mu}_\varepsilon$ [%]				$\boldsymbol{\sigma}_\varepsilon$ [%]				$\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ [%]			
		d_1	d_2	k_1	k_2	d_1	d_2	k_1	k_2	d_1	d_2	k_1	k_2
TS (dense)	510	-0.290	0.441	-0.233	-0.070	2.845	2.525	1.710	0.715	2.227	2.030	1.307	0.562
TS (sparse)	34	0.830	0.152	-0.699	-0.431	4.102	4.049	1.518	0.657	3.248	3.230	1.296	0.623
TE	12	1.817	-0.047	1.659	-0.103	11.013	3.477	5.316	1.434	9.232	2.749	4.375	1.138
MP	512	1.945	0.947	-1.386	-0.337	11.810	7.642	7.106	3.020	9.778	6.453	5.675	2.407
RI	512	0.280	0.780	-0.907	-0.946	6.291	7.482	2.542	1.583	4.981	6.083	2.076	1.451
MP (FBoI)	34	0.421	0.271	0.429	0.200	3.561	2.927	1.723	0.572	2.735	2.262	1.256	0.453
RI (FBoI)	34	-0.213	-0.450	0.013	-0.188	2.699	2.555	1.145	0.515	2.129	2.062	0.885	0.427

4.2.2 Updated model accuracy

As shown in the previous section, the IMPU method is able to infer parameter values with average relative absolute errors of approximately 2% or lower. However, one of the main purposes of parameter updating is to obtain a more accurate model. Therefore, the accuracy of the updated model, i.e., the reference model parametrized with the inferred updating parameter values, is briefly discussed here.

For this purpose, a single test sample is selected of which the true parameter estimates $\bar{\boldsymbol{p}}$ are listed in Table 6. Simulating a (noise injected) measurement for the system parameterized with these true parameter values results in the solid blue line in Figure 14. For this simulation, we have used the same (simulated) experiment settings as discussed in Section 4.1.1. Extracting RI (FBoI) features from this signal and using the computationally cheap IMPU method to infer 'updated' parameter values $\hat{\boldsymbol{p}}$ yields the values listed in Table 6. Simulating the reference model, parameterized with these updated parameter values, yields the dashed orange line in Figure 14. Here, we see that, for both DoFs, there is high agreement in the measured and updated response. Furthermore, the response of the reference model (with an initial guess for the updating parameter values $\boldsymbol{p} = \boldsymbol{p}_c$) is plotted (dashed purple line) clearly demonstrating the benefit of updating the model parameters.

Due to the retained model structure, the updated model is also accurate for (simulated) experiments with different initial conditions and excitation signals (as may be expected). For example, Figure 15 shows the response of the, among others, updated model when using a harmonic excitation signal, i.e., $u_1(t_k) = 0.15\cos(\omega_1(\boldsymbol{p}_c)t_k)$ and $u_2(t_k) = 0.01\cos(\omega_2(\boldsymbol{p}_c)t_k)$, and deviating initial conditions, i.e., $\boldsymbol{y}(t_1 = 0) = \begin{bmatrix}0.055 & 0.5\end{bmatrix}^\top$ and $\dot{\boldsymbol{y}}(t_1 = 0) = \begin{bmatrix}-0.01 & 2\end{bmatrix}^\top$. Here, $\omega_1(\boldsymbol{p}_c)$ and $\omega_2(\boldsymbol{p}_c)$ represent the damped angular eigenfrequencies of the linearized (reference) model parameterized with $\boldsymbol{p} = \boldsymbol{p}_c$.

Table 6 Parameter values used to parameterize models used in Figures 14 and 15.

Parameter	d_1 $[\frac{\text{N}\cdot\text{s}}{\text{m}}]$	d_2 $[\frac{\text{N}\cdot\text{m}\cdot\text{s}}{\text{rad}}]$	k_1 $[\frac{\text{N}}{\text{m}}]$	k_2 $[\frac{\text{N}\cdot\text{m}}{\text{rad}}]$
$\bar{\boldsymbol{p}}$	1.082	1.886×10^{-4}	11.51	3.919×10^{-2}
$\hat{\boldsymbol{p}}$	1.097	1.891×10^{-4}	11.38	3.908×10^{-2}
$\boldsymbol{p}_c$	1.000	2.000×10^{-4}	10.00	3.600×10^{-2}

4.3 Generalization capabilities

Since the IMM is learned using training data, it is important to obtain insight in how accurate the IMM is for data it has not yet seen before. Note that the analysis in Section 4.2.1, which is similar to the analysis in [16], hardly gives insight in the generalization capabilities of the ANN. Therefore, in this section, generalization is analyzed by 1) varying the number of training samples, and 2) excluding parts of the training samples.

4.3.1 Number of training samples

The impact of the number of training samples, n_s, on the accuracy of the inferred parameters is analyzed, such that insight in the generalization capabilities of the discussed output features is obtained. Here, the training and testing space are equivalent and as defined in Table 1. The mean absolute relative error for each updating parameter has been plotted versus n_s in Figure 16, for the different output feature types.

In general, for $n_s < 100$, the accuracy of the parameter estimates is observed to be relatively low, showing that these low amounts of

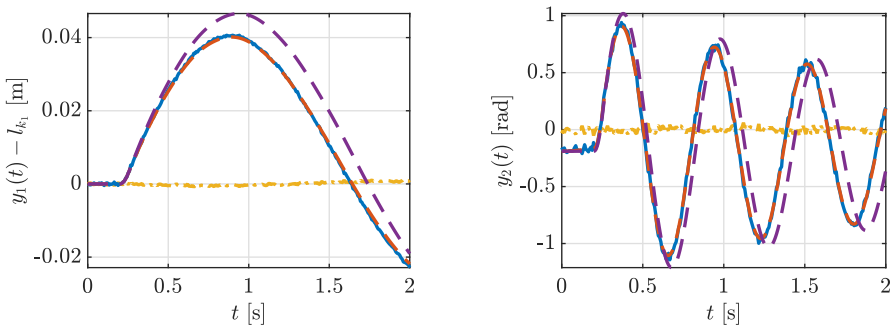

Fig. 14 Comparison of simulated measurement (——) and simulation of updated model (- - -), for the (simulated) experiment used to train the ANN, i.e., impulse-like excitation and the static equilibrium of the reference system with $\boldsymbol{p} = \boldsymbol{p}_c$ as initial conditions. Furthermore, the error of the updated model (-·-) is plotted and, for reference, the simulation is performed using the reference model with $\boldsymbol{p} = \boldsymbol{p}_c$ (- - -).

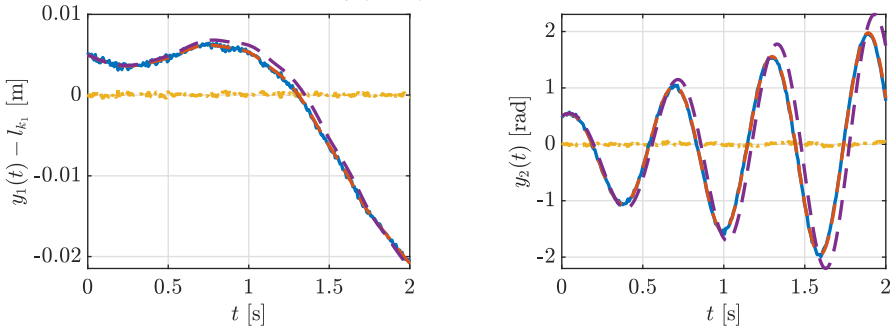

Fig. 15 Comparison of simulated measurement (——) and simulation of updated model (- - -), for different (simulated) experiment (harmonic excitation and deviating initial conditions) than used to update the model. Furthermore, the error of the updated model (-·-) is plotted and, for reference, the simulation is performed using the reference model with $\boldsymbol{p} = \boldsymbol{p}_c$ (- - -).

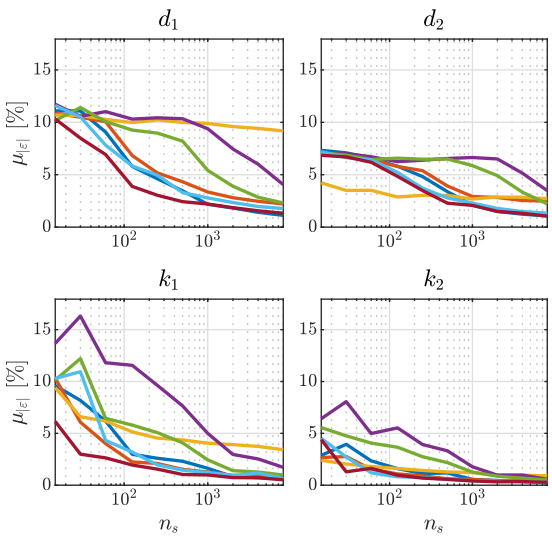

Fig. 16 Mean absolute relative error as a function of the number of training samples, n_s, per updating parameter, for the different output feature types: TS (dense) (——), TS (sparse) (——), TE (——), MP (——), RI (——), MP (FBoI) (——), and RI (FBoI) (——).

training samples do not suffice to properly learn the inverse relation between output features and parameter values. It should be noted, however, that the updating parameter space is, in this case, four-dimensional. Consequently, such few training samples do not sufficiently cover $\mathbb{P}$ and hence performance is poor. Overall, it seems that, for almost all analyzed values of n_s, the RI (FBoI) features perform best or approximately equally as good as the next best alternative(s). Furthermore, when using all frequency bins for the MP and RI features, significantly more training samples are required to achieve the same accuracy as when only information in the FBoIs is used. Again, this is caused by the large amount of redundant and noisy high frequency information, making these features generalize relatively poorly. Since all TS features in both the dense and sparse variant carry useful information about the system, we do not see this behavior for the time-series-based features. Finally, the accuracy of the time extrema features is relatively constant, although comparatively low,

when varying n_s (especially for d_1 and d_2). This shows that the relationship between the values of the damping parameters and the output features is relatively easy to learn for the ANN, i.e., TE features generalize well for different values of n_s. This agrees with the fact that the difference in magnitude between subsequent maxima/minima is closely connected to the damping values.

4.3.2 Excluded training zones

In this section, generalization is analyzed by deliberately omitting parts of the training space and testing how accurate test samples are in distinct 'testzones'. In the following, changes are only made to the subspace of $\mathbb{P}$ spanned by d_1 and d_2. Specifically, for both d_1 and d_2, the test subspace is extended on both sides of the training space with 50% of its width. Furthermore, for both damping parameters, a band, centered in the d_1-d_2 subspace, with a width of 20% of the training subspace width is excluded from the training data. In Figure 17, all test samples are allocated to six testzones, indicated by a color-coded box, where all retained training samples are located in testzone 1. A higher number of a testzone indicates decreasing number and proximity of adjacent boxes with training data (or zone 1 boxes). For example, each box in zone 2 shares two edges with adjacent zone 1 boxes and for zone 6 only one of the corners of each box coincides with the corner of a zone 1 box.

To acquire the training data, the LHC sampled training data used in Section 4 is reused, of which the samples located in testzones 2 and 3 are excluded. Consequently, as we started with 1000 training samples (as used for Figure 17), after exclusion approximately $n_s = 645$ retained training samples are left. Due to the exclusion of samples, the 'density' of training samples per test sample is kept identical in testzone 1 with respect to the analysis in Section 4.2. The validation parameter space (here only used for early stopping) is obtained in an equivalent manner as the training parameter space.

To analyze the effect of these changes in the training and testing parameter subspaces, firstly we focus on the case with $n_s = 645$ obtained using RI (FBoI) features. The absolute relative error per parameter is plotted for each test sample in the d_1-d_2 subspace in Figure 17. In the upper left plot of Figure 17, we observe that for low and high values of d_1 (near the edges of the test subspace), the absolute relative error of d_1 increases. This is simply explained by the fact that the true values for d_1 are far away from the training data, forcing the ANN to extrapolate with respect to d_1. Similar observations are made for d_2. Please also note the similarity of this finding with the observation that test samples near the edge of the training space yield relatively poor results, as discussed in Section 4.2.1. In contrast, the accuracy of k_1 and k_2 is hardly affected by changes in the training and test d_1-d_2 subspace (apart from some testsamples in zone 6), as is also seen in Figure 18. This is explained by the fact that alterations in training and testing data are restricted to the d_1-d_2 subspace.

Table 7 Mean absolute relative error (in %) per updating parameter for all feature types, for testzone 1 when training samples are excluded such that $n_s = 645$.

Feature Type	$\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ [%] d_1	d_2	k_1	k_2
TS (dense)	2.884	2.461	1.626	0.771
TS (sparse)	3.838	3.555	1.181	0.611
TE	11.175	2.701	4.388	1.165
MP	11.246	7.564	7.855	2.980
RI	7.486	7.659	3.917	2.044
MP (FBoI)	3.074	2.569	1.435	0.562
RI (FBoI)	2.362	2.347	1.115	0.515

In Table 7, the mean absolute relative errors $\boldsymbol{\mu}_{|\boldsymbol{\varepsilon}|}$ for the test samples in testzone 1 are listed (as calculated using (22), but summed over the test samples in testzone 1 only), for the different feature types. Since the density of training samples per test sample is roughly equivalent, we can compare these metrics to those found in Table 5. As seen, proportionate errors are indeed found: due to the lower number of training samples in total, $\boldsymbol{\mu}_{|\varepsilon|}$ in Table 7 is, however, typically somewhat higher.

In Figure 18, mean absolute relative errors $\boldsymbol{\mu}_{|\varepsilon|}$ for the different testzones (as calculated using (22) but summed over the test samples per evaluated testzone), for both the RI and RI (FBoI) features, are plotted as a function of n_s. Focusing on the $\boldsymbol{\mu}_{|\varepsilon|}$ for d_1 and d_2, we observe that, for most values of n_s, testzone 6 is least accurate, followed, in order, by zones 5, 4, 1, 2, and 3, with testzone 3 being the most accurate. Although this result might, initially, be counterintuitive (one would expect testzone 1 to be most accurate).

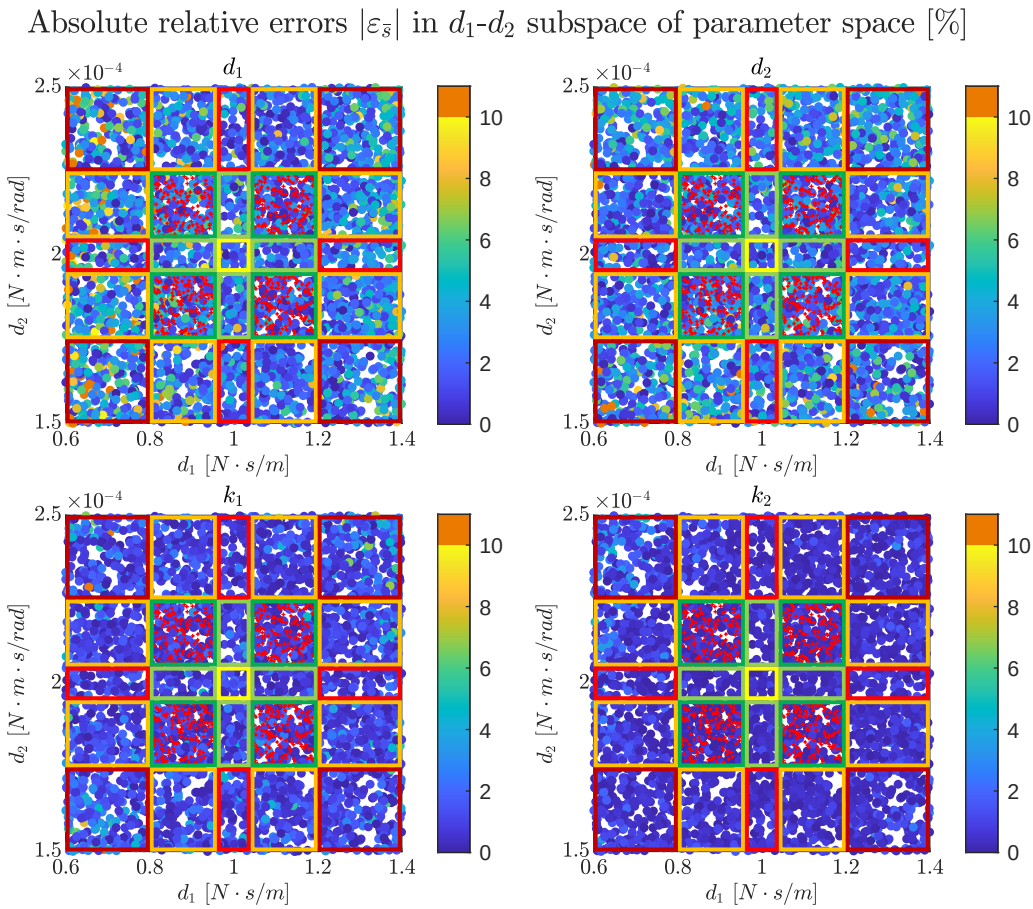

Fig. 17 Absolute relative error (in %) of each test sample $|\varepsilon|$ (see (19) for definition of ε) in the d_1-d_2 subspace of $\mathbb{P}$ per updating parameter, as obtained by using the RI (FBoI) output features with $n_s = 645$. Red plus signs (+) indicate locations of training data samples. Testzones are indicated by the color-coded rectangles as follows: zone 1 (■), zone 2 (■), zone 3 (■), zone 4 (■), zone 5 (■), and zone 6 (■). Note that all absolute relative errors larger than 10% are colored orange.

This observation is, again, closely connected to an aforementioned observation in Section 4.2.1. Namely, the accuracy within a zone is affected by the relative number of test samples of that zone that lie close to the outer ranges of the training subspace. For example, each area belonging to testzone 1 has two relatively long edges beyond which no training samples are present. In contrast, the test samples in testzone 3 lie in the center between the lowest and highest training values of d_1 and d_2. For (approximately) $n_s > 2500$, this order of accuracy does not hold anymore, since then this phenomenon is alleviated due to the large presence of training samples overall.

Furthermore, as is to be expected, Figure 18 shows the trend that $\boldsymbol{\mu}_{|\varepsilon|}$ decreases with n_s for all testzones. Consequently, as each application requires a different level of accuracy, the number of training samples may be used to tune the accuracy of the IMPU method in an a posteriori manner. Note that, although $\boldsymbol{\mu}_{|\varepsilon|}$ seems to approach 0 for large n_s, the influence of measurement noise and the inexact nature of the trained IMM will always cause some error in the inferred parameter values.

Additionally, comparing the results obtained using the RI and RI (FBoI) features, we see that, again, the RI (FBoI) features outperform the full frequency content RI features, where the latter requires relatively many training samples to yield comparable accuracy. Note that, for, e.g., $n_s = 645$, the RI (FBoI) features are better at extrapolating outside the outer ranges of the training space, i.e., zones 4, 5, and 6, than the RI features are at interpolating, i.e., zones 1, 2, and

3. Furthermore, the qualitative behavior of the results, i.e., high errors for low n_s that decrease for increasing values of n_s, is similar for RI features and RI (FBoI) features.

To conclude, in general, if sufficient training data is used, even test samples in zones outside the training subspace can be approximated relatively accurately when using RI (FBoI). For example, the mean absolute relative errors, calculated for testzones 1, 3, and 6 are listed for d_1 and d_2 in Tables 8 and 9, respectively. Here, we observe that the values of $\boldsymbol{\mu}_{|\varepsilon|}$ in testzone 6 is, with some exceptions for the TE and MP features, approximately two times higher than the values of $\boldsymbol{\mu}_{|\varepsilon|}$ in testzones 1 and 3. Given the fact that only few training samples lie somewhat close to testzone 6, this is an acceptable result. Due to the already high relative errors for the TE, MP, and RI features in testzones 1 and 3, their relative errors in testzone 6 are, however, regarded as too high.

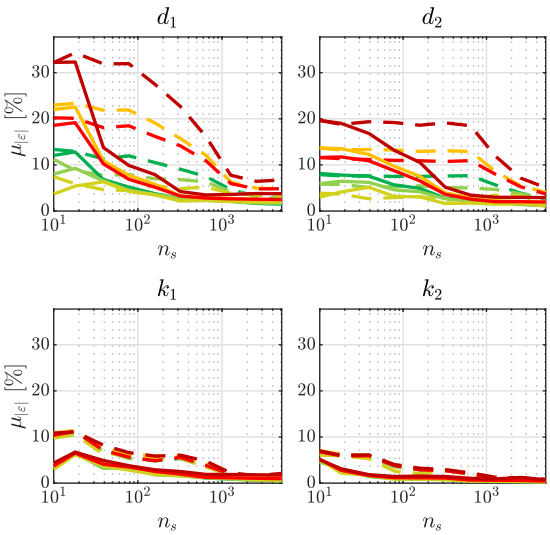

Fig. 18 Mean absolute relative error (in %) per testzone as a function of the number of training samples, n_s, per updating parameter. The results indicated with solid lines (——) are obtained using RI (FBoI) features and results indicated with dashed lines (– – –) are obtained using RI features. The colors of the solid/dashed lines correspond to testzones 1 (——), 2 (——), 3 (——), 4 (——), 5 (——), and 6 (——), respectively.

Table 8 Mean absolute relative error of d_1 estimate for testzones 1, 3, and 6, obtained using $n_s = 2569$ training samples.

Feature Type	$\boldsymbol{\mu}_{\|\varepsilon\|}$ of d_1 [%] Testzone 1	Testzone 3	Testzone 6
TS (dense)	1.778	1.644	4.390
TS (sparse)	2.779	2.590	5.929
TE	10.996	5.141	28.593
MP	5.849	5.827	12.606
RI	3.209	3.435	6.437
MP (FBoI)	2.131	2.299	4.865
RI (FBoI)	1.661	1.877	3.790

Table 9 Mean absolute relative error of d_2 estimate for testzones 1, 3, and 6, obtained using $n_s = 2569$ training samples.

Feature Type	$\boldsymbol{\mu}_{\|\varepsilon\|}$ of d_2 [%] Testzone 1	Testzone 3	Testzone 6
TS (dense)	1.372	1.156	2.415
TS (sparse)	2.627	2.813	4.921
TE	2.583	3.138	7.067
MP	5.396	4.262	11.015
RI	3.893	3.293	7.139
MP (FBoI)	1.606	1.759	3.007
RI (FBoI)	1.443	1.544	2.963

5 Conclusions and future work

To minimize the mismatch between a physical system and its model (or digital twin), this paper proposes to update physically interpretable parameter values of nonlinear dynamical models in real-time by an Inverse Mapping Parameter Updating (IMPU) approach. This approach consists of an offline and an online phase. In the offline phase, an Artificial Neural Network (ANN) is trained based on simulated response features. In the offline phase, the ANN is used to map measured transient-based response features to inferred parameter estimates. For this purpose, novel transient-based feature types have been introduced and evaluated on a (simulated) 2-DoF dynamical multibody system. Engineering knowledge about the system is used to define informative output features resulting in improved accuracy and computational efficiency of the IMPU approach. Additionally, generalization capabilities are improved by properly choosing output features such that relatively few training samples already achieve satisfactory accuracy and parameter values outside the training space are estimated with acceptable accuracy.

For future work, the authors intend to apply the introduced methodology to a physical system.

Acknowledgements This publication is part of the project Digital Twin [67] (project 2.1) with project number P18-03 of the research programme Perspectief which is (mainly) financed by the Dutch Research Council (NWO).

Declarations

- **Funding**, This work was (mainly) financed by the Dutch Research Council (NWO) as part of the Digital Twin project (subproject 2.1) with number P18-03 in the research programme Perspectief.
- **Competing Interests**, The authors have no relevant financial or non-financial interests to disclose.
- **Author contributions**, All authors contributed to the study conception and design. Material preparation, data collection, analysis, and writing of the first draft were performed by Bas Kessels and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.
- **Data availability**, As the model and (simulated) experiments used to generate all training/test/validation data has been described in detail, simulations to obtain these data sets can be easily reproduced. Therefore, and due to the sheer size of this data, the data sets are not made directly downloadable. Upon special, reasonable request, the corresponding author may be approached such that (parts of) the data sets can be shared.

Appendix A Normalization of parameter values and features

As discussed in Section 2.2.2, prior to being used in the ANN, both the features and parameters are normalized to increase robustness of the ANN. In the following, the normalization procedure is explained for training parameter vector $\boldsymbol{p}_s$. However, the presented procedure is equivalently applicable to training feature vector $\boldsymbol{\psi}(\boldsymbol{p}_s)$.

Entry $j \in \{1, \ldots, n_p\}$ of $\boldsymbol{p}_s$ is denoted by $\boldsymbol{p}_s[j]$. The normalization is performed such that the normalized value, $\boldsymbol{p}_s^{\text{nor}}[j]$, equals 0 for the lowest valued entry across all n_s training samples and equals 1 for the highest valued entry across all training samples:

$$\boldsymbol{p}_s^{\text{nor}}[j] = \frac{\boldsymbol{p}_s[j] - \boldsymbol{p}_{\min}[j]}{\boldsymbol{p}_{\max}[j] - \boldsymbol{p}_{\min}[j]}, \tag{A1}$$

where the smallest and highest values of $\boldsymbol{p}_s[j]$ across all training samples are given by $\boldsymbol{p}_{\min}[j]$ and $\boldsymbol{p}_{\max}[j]$, respectively:

$$\boldsymbol{p}_{\min}[j] = \min\left(\boldsymbol{p}_1[j], \ldots, \boldsymbol{p}_{n_s}[j]\right), \tag{A2}$$

$$\boldsymbol{p}_{\max}[j] = \max\left(\boldsymbol{p}_1[j], \ldots, \boldsymbol{p}_{n_s}[j]\right). \tag{A3}$$

For the normalization of measured features in the online phase, these (measured) features are normalized using the normalization bounds $\boldsymbol{\psi}_{\min}$ and $\boldsymbol{\psi}_{\max}$ as obtained using equivalent variants of (A2) and (A3) for the feature vectors:

$$\bar{\boldsymbol{\psi}}_{\bar{s}}^{\text{nor}}[j] = \frac{\bar{\boldsymbol{\psi}}_{\bar{s}}[j] - \boldsymbol{\psi}_{\min}[j]}{\boldsymbol{\psi}_{\max}[j] - \boldsymbol{\psi}_{\min}[j]}. \tag{A4}$$

Here, it is emphasized that $\boldsymbol{\psi}_{\min}$ and $\boldsymbol{\psi}_{\max}$ are thus obtained using training data.

To calculate the non-normalized parameter values using the normalized parameter values as inferred by the ANN, we again use the normalization bounds calculated in the training phase:

$$\hat{\boldsymbol{p}}_{\bar{s}}[j] = \boldsymbol{p}_{\min}[j] + \hat{\boldsymbol{p}}_{\bar{s}}^{\text{nor}}[j]\left(\boldsymbol{p}_{\max}[j] - \boldsymbol{p}_{\min}[j]\right). \tag{A5}$$

Note that validation samples are normalized in an equivalent manner as the measured (or testing) samples, i.e., with normalization bounds calculated in the training phase.

Appendix B Equations of motion

The EoMs of the demonstrator system introduced in Section 4.1.1 are

$$\boldsymbol{M}(\boldsymbol{y})\ddot{\boldsymbol{y}} + \boldsymbol{c}(\boldsymbol{y}, \dot{\boldsymbol{y}}) + \boldsymbol{g}(\boldsymbol{y}) + \boldsymbol{f}(\boldsymbol{y}, \dot{\boldsymbol{y}}) = \boldsymbol{u}, \tag{B6}$$

where explicit dependency on t and $\boldsymbol{p}$ are omitted for brevity. Furthermore, the mass matrix $\boldsymbol{M}$, the vector containing Coriolis and centripetal forces $\boldsymbol{c}$, the vector with gravitational forces $\boldsymbol{g}$, and the vector with spring and damper forces $\boldsymbol{f}$ are given

by

$$\begin{aligned}
\boldsymbol{M}(\boldsymbol{y}) &= \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \\
\boldsymbol{c}(\boldsymbol{y}, \dot{\boldsymbol{y}}) &= \begin{bmatrix} -m_2 \dot{y}_2^2 (l_{\mathrm{CoM},1} \cos(y_2) - l_{\mathrm{CoM},2} \sin(y_2)) \\ 0 \end{bmatrix} \\
\boldsymbol{g}(\boldsymbol{y}) &= \begin{bmatrix} 0 \\ m_2 g l_{\mathrm{CoM},1} \cos(y_2) - m_2 g l_{\mathrm{CoM},2} \sin(y_2) \end{bmatrix} \\
\boldsymbol{f}(\boldsymbol{y}, \dot{\boldsymbol{y}}) &= \begin{bmatrix} k_1(y_1 - l_{k1}) + d_1 \dot{y}_1 \\ k_2(y_2 - \theta_{k2}) + d_2 \dot{y}_2 \end{bmatrix},
\end{aligned} \tag{B7}$$

where

$$\begin{aligned}
&M_{11} = m_1 + m_2, \\
&M_{12} = M_{21} = -m_2(l_{\mathrm{CoM},1} \sin(y_2) + l_{\mathrm{CoM},2} \cos(y_2)), \\
&M_{22} = m_2 l_{\mathrm{CoM},1}^2 + m_2 l_{\mathrm{CoM},2}^2 + I_{\mathrm{CoM},2}.
\end{aligned}$$

Appendix C Linearized system properties

The Frequency Response Functions (FRFs) of the academic demonstrator is plotted in Figure C1 for each parameter case, as defined in Table 3. From the FRFs it is observed that, for this system, the DoFs are largely uncoupled, i.e., y_i is mostly influenced by u_i, with i indicating the DoF. This is also apparent from the eigenmodes $\boldsymbol{\phi}_i$ of parameter case 1:

$$\begin{aligned}
\boldsymbol{\phi}_1 &= \begin{bmatrix} 1 \\ -3.80 \times 10^{-2} - 7.88 \times 10^{-3} j \end{bmatrix} \\
\boldsymbol{\phi}_2 &= \begin{bmatrix} 1.29 \times 10^{-4} + 5.23 \times 10^{-6} j \\ 1 \end{bmatrix},
\end{aligned} \tag{C8}$$

with j the imaginary unit.

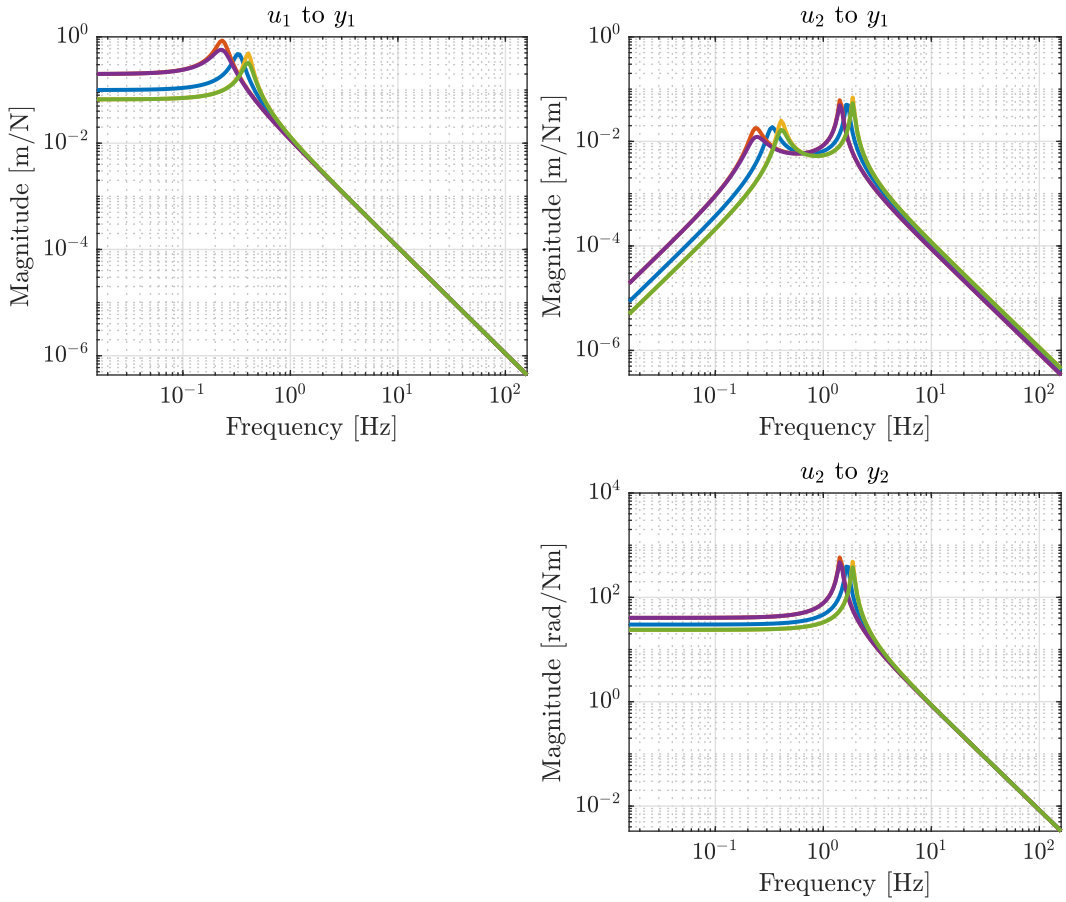

Fig. C1 Frequency response functions for linearized models for the five parameter cases. The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—). Please note that the FRF from u_2 to y_1 is equal to the FRF from u_1 to y_2.

References

[1] Haag, S., Anderl, R.: Digital twin – Proof of concept. Manufacturing Letters **15**, 64–66 (2018). https://doi.org/10.1016/j.mfglet.2018.02.006

[2] Grieves, M., Vickers, J.: Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In: Kahlen, F.J., Flumerfelt, S., Alves, A. (eds.) Transdisciplinary Perspectives on Complex Systems, pp. 85–113. Springer, Cham, Switzerland (2017). https://doi.org/10.1007/978-3-319-38756-7_4

[3] Glaessgen, E.H., Stargel, D.S.: The digital twin paradigm for future NASA and U.S. Air force vehicles. In: Structural Dynamics and Materials, pp. 1–14 (2012). https://doi.org/10.2514/6.2012-1818

[4] Karve, P.M., Guo, Y., Kapusuzoglu, B., Mahadevan, S., Haile, M.A.: Digital twin approach for damage-tolerant mission planning under uncertainty. Engineering Fracture Mechanics **225** (2020). https://doi.org/10.1016/j.engfracmech.2019.106766

[5] Grieves, M.: Digital twin: manufacturing excellence through virtual factory replication. White Paper (2014)

[6] Mottershead, J.E., Friswell, M.I.: Model updating in structural dynamics: A survey. Journal of Sound and Vibration **167**(2), 347–375 (1993)

[7] Mottershead, J.E., Link, M., Friswell, M.I.: The sensitivity method in finite element model updating: A tutorial. Mechanical Systems and Signal Processing **25**(7), 2275–2296 (2011). https://doi.org/10.1016/j.ymssp.2010.10.012

[8] Kennedy, M.C., O'Hagan, A.: Bayesian calibration of computer models. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **63**(3), 425–464 (2001). https://doi.org/10.1111/1467-9868.00294

[9] Viana, F.A.C., Nascimento, R.G., Dourado, A., Yucesan, Y.A.: Estimating model inadequacy in ordinary differential equations with physics-informed neural networks. Computers and Structures **245**, 106458 (2021). https://doi.org/10.1016/j.compstruc.2020.106458

[10] Chiandussi, G., Bugeda, G., Oñate, E.: A simple method for automatic update of finite element meshes. Communications in Numerical Methods in Engineering **16**(1), 1–19 (2000). https://doi.org/10.1002/(SICI)1099-0887(200001)16:1<1::AID-CNM310>3.0.CO;2-A

[11] Sehgal, S., Kumar, H.: Structural dynamic model updating techniques: A state of the art review. Archives of Computational Methods in Engineering **23**(3), 515–533 (2016). https://doi.org/10.1007/s11831-015-9150-3

[12] Hemez, F.M., Doebling, S.W.: Inversion of structural dynamics simulations: State-of-the-art and orientations of the research. In: International Conference on Noise and Vibration Engineering, vol. 25, pp. 425–435 (2000)

[13] Hemez, F.M., Doebling, S.W.: Review and assessment of model updating for non-linear, transient dynamics. Mechanical Systems and Signal Processing **15**(1), 45–74 (2001). https://doi.org/10.1006/mssp.2000.1351

[14] Li, W., Chen, Y., Lu, Z.R., Liu, J., Wang, L.: Parameter identification of nonlinear structural systems through frequency response sensitivity analysis. Nonlinear Dynamics **104**(4), 3975–3990 (2021). https://doi.org/10.1007/s11071-021-06481-5

[15] Verbeek, G., De Kraker, A., Van Campen, D.H.: Nonlinear parametric identification using periodic equilibrium states - Application to an aircraft landing gear damper. Nonlinear Dynamics **7**(4), 499–515 (1995). https://doi.org/10.1007/BF00121110

[16] Atalla, M.J., Inman, D.J.: On model updating using neural networks. Mechanical Systems and Signal Processing **12**(1), 135–161 (1998). https://doi.org/10.1006/mssp.1997.0138

[17] Diaz, M., Charbonnel, P., Chamoin, L.: Robust energy-based model updating framework for random processes in dynamics: Application to shaking-table experiments. Computers and Structures **264**, 106746 (2022). https://doi.org/10.1016/j.compstruc.2022.106746

[18] Arora, V., Singh, S.P., Kundra, T.K.: Damped model updating using complex updating parameters. Journal of Sound and Vibration **320**(1-2), 438–451 (2009). https://doi.org/10.1016/j.jsv.2008.08.014

[19] Friswell, M.I., Mottershead, J.E., Ahmadian, H.: Finite-element model updating using experimental test data: Parametrization and regularization. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **359**(1778), 169–186 (2001). https://doi.org/10.1098/rsta.2000.0719

[20] Kim, K.O., Anderson, W.J., Sandstrom, R.E.: Nonlinear inverse perturbation method in dynamic analysis. AIAA Journal **21**(9), 1310–1316 (1983). https://doi.org/10.2514/3.8245

[21] Lin, R.M., Zhu, J.: Finite element model updating using vibration test data under base excitation. Journal of Sound and Vibration **303**, 596–613 (2007). https://doi.org/10.1016/j.jsv.2007.01.029

[22] Wang, W., Mottershead, J.E., Ihle, A., Siebert, T., Reinhard Schubach, H.: Finite

element model updating from full-field vibration measurement using digital image correlation. Journal of Sound and Vibration **330**(8), 1599–1620 (2011). https://doi.org/10.1016/j.jsv.2010.10.036

[23] Modak, S.V., Kundra, T.K., Nakra, B.C.: Comparative study of model updating methods using simulated experimental data. Computers and Structures **80**(5-6), 437–447 (2002). https://doi.org/10.1016/S0045-7949(02)00017-2

[24] Sidhu, J., Ewins, D.J.: Correlation of finite element and modal testing studies of a practical structure. In: 2nd International Modal Analysis Conference, Orlando, USA (1984)

[25] Berman, A., Nagy, E.J.: Improvement of a large analytical model using test data. AIAA Journal **21**(8), 1168–1173 (1983)

[26] Caesar, B.: Updating system matrices using modal test data. In: 5th International Modal Analysis Conference, London, England (1987)

[27] Åström, K.J., Eykhoff, P.: System identification - A survey. Automatica **7**(2), 123–162 (1971). https://doi.org/10.1016/0005-1098(71)90059-8

[28] Ljung, L.: System Identification - Theory for the User, 2nd edn. Pearson, Linköping (1997)

[29] Pintelon, R., Schoukens, J.: System Identification: A Frequency Approach, 2nd edn. Wiley, Piscataway (2012)

[30] Kerschen, G., Worden, K., Vakakis, A.F., Golinval, J.C.: Nonlinear system identification in structural dynamics: Current status and future directions. In: Society for Experimental Mechanics (2007)

[31] Schoukens, J., Ljung, L.: Nonlinear system identification: A user-oriented road map. IEEE Control Systems **39**(6), 28–99 (2019) arXiv:1902.00683. https://doi.org/10.1109/MCS.2019.2938121

[32] Berman, A.: System identification of structural dynamic models - Theoretical and practical bounds. In: 25th Structures, Structural Dynamics and Materials Conference. American Institute of Aeronautics and Astronautics, Reston, Virigina (1984). https://doi.org/10.2514/6.1984-929. http://arc.aiaa.org/doi/10.2514/6.1984-929

[33] Li, S., Yang, Y.: Data-driven identification of nonlinear normal modes via physics-integrated deep learning. Nonlinear Dynamics **106**(4), 3231–3246 (2021). https://doi.org/10.1007/s11071-021-06931-0

[34] Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering **360** (2020). https://doi.org/10.1016/j.cma.2019.112789

[35] Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045

[36] Zhang, E., Yin, M., Karniadakis, G.E.: Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging. arXiv (2020) arXiv:2009.04525

[37] Yan, C.A., Vescovini, R., Dozio, L.: A framework based on physics-informed neural networks and extreme learning for the analysis of composite structures. Computers & Structures **265**, 106761 (2022). https://doi.org/10.1016/j.compstruc.2022.106761

[38] Brunton, S.L., Proctor, J.L., Kutz, J.N., Bialek, W.: Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proceedings of the National Academy of Sciences of the United States of America **113**(15), 3932–3937 (2016) arXiv:1509.03580. https://doi.org/10.1073/pnas.1517384113

[39] Kaiser, E., Kutz, J.N., Brunton, S.L.: Sparse identification of nonlinear dynamics for

model predictive control in the low-data limit. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **474**(2219) (2018) arXiv:1711.05501. https://doi.org/10.1098/rspa.2018.0335

[40] Quade, M., Abel, M., Kutz, J.N., Brunton, S.L.: Sparse identification of nonlinear dynamics for rapid model recovery. Chaos **28**(6) (2018) arXiv:1803.00894. https://doi.org/10.1063/1.5027470

[41] Champion, K., Lusch, B., Kutz, J.N., Brunton, S.L.: Data-driven discovery of coordinates and governing equations. Proceedings of the National Academy of Sciences of the United States of America **116**(45), 22445–22451 (2019) arXiv:1904.02107. https://doi.org/10.1073/pnas.1906995116

[42] Kapteyn, M.G., Knezevic, D.J., Willcox, K.: Toward predictive digital twins via component-based reduced-order models and interpretable machine learning. In: AIAA Scitech Forum & Exhibition (2020). https://doi.org/10.2514/6.2020-0418

[43] Kapteyn, M.G., Knezevic, D.J., Huynh, D.B.P., Tran, M., Willcox, K.E.: Data-driven physics-based digital twins via a library of component-based reduced-order models. International Journal for Numerical Methods in Engineering (May), 1–18 (2020). https://doi.org/10.1002/nme.6423

[44] Kapteyn, M.G., Willcox, K.E.: From physics-based models to predictive digital twins via interpretable machine learning. arXiv (2020) arXiv:2004.11356

[45] Lecerf, M., Allaire, D., Willcox, K.: Methodology for dynamic data-driven online flight capability estimation. AIAA Journal **53**(10), 3073–3087 (2015). https://doi.org/10.2514/1.J053893

[46] Singh, V., Willcox, K.E.: Methodology for path planning with dynamic data-driven flight capability estimation. 17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 1–22 (2016). https://doi.org/10.2514/6.2016-4124

[47] Levin, R.I., Lieven, N.A.J.: Dynamic finite element model updating using neural networks. Journal of Sound and Vibration **210**(5), 593–607 (1998)

[48] Yong, L., Zhenguo, T.: A two-level neural network approach for dynamic FE model updating including damping. Journal of Sound and Vibration **275**(3-5), 931–952 (2004). https://doi.org/10.1016/S0022-460X(03)00796-X

[49] Miller, B.: Application of neural networks for structure updating. Computer Assisted Mechanics and Engineering Sciences **18**(3), 191–203 (2011)

[50] Neri, R., Arras, M., Coppotelli, G.: FRF-based model updating using neural networks. In: ISMA, pp. 3243–3258 (2016)

[51] Zimmerman, D.C., Hasselman, T., Anderson, M.: Approximation and identification of nonlinear structural dynamics. Nonlinear Dynamics **39**, 113–128 (2005)

[52] Kessels, B.M., Korver, J.N., Fey, R.H.B., van de Wouw, N.: Model updating for digital twins using Gaussian process inverse mapping models. In: ENOC 2020+2 (July 18-22, 2022), Lyon, France (accepted for publication) (2022). https://doi.org/10.1007/978-3-031-04122-8_1

[53] Kessels, B.M., Fey, R.H.B., Abbasi, M.H., van de Wouw, N.: Model updating for nonlinear dynamic digital twins using data-based inverse mapping models. In: IMAC-XL, A Conference and Exposition on Structural Dynamics 2022. Springer, Cham, Switzerland (2022)

[54] Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Cambridge (2006)

[55] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)

[56] Reed, R., Marks, R.J.: Neural Smithing. The MIT Press, Cambridge (1999). https://doi.

org/10.7551/mitpress/4937.001.0001

[57] McKay, M.D., Beckman, R.J., Conover, W.J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics **21**(2), 239–245 (1979). https://doi.org/10.2307/1268522

[58] Prechelt, L.: Early stopping - But when. In: Neural Networks Tricks of the Trade, pp. 55–70 (1998)

[59] MathWorks: Matlab findpeaks function documentation. https://nl.mathworks.com/help/signal/ref/findpeaks.html Accessed 2022-07-09

[60] Keesman, K.J., Stigter, J.D.: Optimal parametric sensitivity control for the estimation of kinetic parameters in bioreactors. Mathematical Biosciences **179**(1), 95–111 (2002). https://doi.org/10.1016/S0025-5564(02)00097-4

[61] Keesman, K.J., Walter, E.: Optimal input design for model discrimination using Pontryagin's maximum principle: Application to kinetic model structures. Automatica **50**(5), 1535–1538 (2014). https://doi.org/10.1016/j.automatica.2014.03.022

[62] Rakin, A.S., He, Z., Fan, D.: Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition **June-2019**, 588–597 (2019) arXiv:1811.09310. https://doi.org/10.1109/CVPR.2019.00068

[63] Zhu, X., Hu, R., Lei, C., Thung, K.H., Zheng, W., Wang, C.: Low-rank hypergraph feature selection for multi-output regression. World Wide Web **22**(2), 517–531 (2019). https://doi.org/10.1007/s11280-017-0514-5

[64] Keras: Adam. https://keras.io/api/optimizers/adam/ Accessed 2022-06-23

[65] Navidi, W.: Statistics for Engineers and Scientists, 3rd edn. McGraw-hill, New York (2006)

[66] Huang, P., Yang, X.: Unsupervised feature selection via adaptive graph and dependency score. Pattern Recognition **127**, 108622 (2022). https://doi.org/10.1016/j.patcog.2022.108622

[67] Digital Twin (2022). https://www.digital-twin-research.nl/ Accessed 2022-06-23