Nonlinear Dyn (2023) 111:10255–10285
https://doi.org/10.1007/s11071-023-08354-5

ORIGINAL PAPER

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

**Bas M. Kessels · Rob H. B. Fey · Nathan van de Wouw**

Received: 31 August 2022 / Accepted: 21 February 2023 / Published online: 26 March 2023


**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

B. M. Kessels (✉) · R. H. B. Fey · N. van de Wouw
Mechanical Engineering, Eindhoven University of Technology, Den Dolech 2, Eindhoven 5612 AZ, The Netherlands
e-mail: b.m.kessels@tue.nl

R. H. B. Fey
e-mail: r.h.b.fey@tue.nl

N. van de Wouw
e-mail: n.v.d.wouw@tue.nl

## 1 Introduction

As part of the fourth industrial revolution, Digital twins (DTs) 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 DT, 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 cur-

rent system, enabling improved design of the next generation. Additionally, measured behavior of the actual system can be compared to behavior predicted by a DT 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 in the lifetime of the system. Furthermore, during the operation of a system, a DT 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 DT 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 DTs 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 [6]. In this paper, these challenges are regarded as out of scope, however, and the assumption is made that a DT is given by a parametric (dynamic) model, derived using either: (1) analytical approaches based on first-principles (FP) such as Lagrange's equations of motion or Hamilton's principle, (2) (multiphysical) modeling and analysis software packages, such as finite element (FE) or multibody dynamics packages, and (3) software tools that combine techniques from multiple engineering disciplines, e.g., dynamic modeling and motion control, such as the Matlab-based Simulink or Simscape packages. Note that, although these three modeling approaches are all based on FPs, only the first approach provides direct access to the closed-form equations of motion (EoMS) for the user. Models from the second and third approach are also referred to as simulation models. But also in the latter two approaches, the user has access to the parameter values. Another challenge is related to the ever-present mismatch between measurements of a physical system and corresponding predictions of its DT. For the DT to reach its full potential, this mismatch should be minimal throughout the system's operational life [1,4,7]. This challenge of giving DTs life-long learning capabilities is referred to as autonomous model updating and is considered in this paper.

In Mottershead et al. [8,9], the aforementioned mismatch between measurement and model predictions is contributed to three distinct model error sources; incorrect model structure [10,11], discretization errors [12], 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. Furthermore, in this work, experimental errors caused by, e.g., sensing or calibration errors, are assumed to be corrected, negligible, or at least to be much smaller than modeling errors, which is the case in many situations.

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, (3) updating should be applicable to (complex) nonlinear dynamical models, and (4) the updating method should be suitable for parametric models (not necessarily of dynamical systems) derived using FPs, FE/multibody software, or multidisciplinary software tools. The presentation of an autonomous model updating method simultaneously fulfilling these four 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 four types. The first type concerns updating methods originating from the field of structural dynamics, see, e.g., [9,13] for an overview. Although some exceptions exist, see [14–17], these updating methods are typically limited to linear(ized) systems [8,18]. Furthermore, these methods are based on expensive iterative sensitivity methods [16,17,19–25] or physically non-interpretable direct methods [21,26–28]. Consequently, these techniques do not meet the four requirements for model updating for DTs as mentioned above and are thus 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, [29–31]. Note that, in contrast to model updating in the field of, typically linear(ized), structural dynamics, system identification has been developed for nonlinear systems [32,33]. 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 [34] and thus, do not satisfy the above requirements.

A third type of parameter updating techniques concerns filter-based (or observer-based) methods. Examples are the vanilla [35], extended [36,37], and unscented Kalman filter [38–40], Gaussian filters [41, 42], and particle filters [43]. Although these approaches are often used for DTs, for complex systems (i.e., systems with many states and/or parameters), filter methods might suffer from relatively high computational costs, prohibiting online updating (especially if the required sampling frequency is relatively high) [43]. Furthermore, filter-based methods require an initial guess for the parameter estimates. If this information is not available or sudden changes to the parameter values occur, filter methods might end up in local minima. To avoid this, initial parameter values may be estimated using a coarse global parameter estimation method [38] at a significant cost of computation time. Finally, filter-based updating methods typically require direct accessibility to closed-form EoMs of the dynamical model for the user. Consequently, filter-based methods cannot be directly applied to simulation models (as obtained from, e.g., FE/multibody software packages, or Matlab-based packages as Simulink or Simscape) that may be used to constitute a digital twin.

The fourth approach to model updating employs machine learning (ML). Here, again, we can distinguish between, firstly, identification, i.e., identifying a system without using a FP 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 [44].

An other example of ML-based identification is the work by Karniadakis et al. [45–47] on physics-informed neural networks (PINNs) that is also adopted by other researchers [48]. In this method, coefficients of (partial) differential equations governing the measured system are learned using a neural network. In general, the learned coefficients of these (partial) differential equations, however, do not refer to directly interpretable physical quantities, e.g., masses, damping, and stiffness constants. Additionally, to update a set of coefficients using measured data, a neural network needs to be trained, which can take significant computation time and hence, impede real-time updating. This holds especially for models consisting of a large number of states and/or parameters. An other ML-based identification methodology is the sparse identification of nonlinear dynamics (SINDy) algorithm developed by Brunton et al. [49–52]. Similar to PINNs, this method finds (a sparse set of) coefficients of differential equations, which may not correspond to physical quantities, again hindering physical interpretation. Note that this argument holds for most system identification methods (based on ML or not). A ML-based method more closely related to (physically interpretable) model updating is found in the work of Willcox et al. [53–57]. Here, a (FP) 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, and consequently is of limited accuracy. Other ML-based methodologies aim at finding, online, the (global) parameter values corresponding to a measured signal (in a continuous domain) using, for example, iterative genetic algorithms [58,59]. In this case, however, the (genetic) search over the parameter space takes considerable computational effort, rendering real-time updating impossible [38].

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 directly maps, with small computational costs, a set of measured features, describing the (dynamical) behavior of a system, to a set of corresponding physically-interpretable parameter values of the reference model. In the literature, (trained) Artificial Neural Networks (ANNs) have been found suitable to constitute these IMMs due to their generic function approximation characteristics [18,60–63]. Consequently, this methodology was coined the neural network updating method (NNUM) by Atalla [18]. 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, 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 [64]. The main qualitative advantage of this method with respect to previously discussed parameter updating methods is that online parameter updating is enabled even for large-scale systems with high sampling frequencies. This is enabled by shifting a significant majority of the computation time to an offline (training) phase, whereas the required online (inference) computations are extremely efficient and scale marginally with the complexity of the system. Although this IMPU methodology thus satisfies requirements 1 (real-time updating) and 2 (physically interpretable updating) for a broad range of systems and their DTs, this IMPU methodology has, to the best of the authors' knowledge, however, only been applied to linear(ized) dynamical systems. Also note that the IMPU method can be applied on parametric models derived using any of the aforementioned model derivation options (first-principles, FE/multibody software, or multi-disciplinary software packages), yielding valuable flexibility in the modeling of the DT.

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 two simulated nonlinear multibody systems.
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 [65], the IMPU method has been applied to update a nonlinear dynamical model. However, this extended abstract only discussed the use of a single, straightforward time-domain-based feature type (samples of measured time-series data as directly obtained from measurements). With respect to this extended abstract, one of the important novelties of this paper lies in the proposal of different transient-based (both time-domain and frequency-domain) feature types that aim at further improving the accuracy of the parameter estimates using engineering knowledge. The current paper expands upon [65] with the following contributions: (1) a full-fledged description of the IMPU method for nonlinear systems, (2) a more complete application and analysis on two simulated multibody systems, one of them including a comparison with the well-known Kalman filtering technique, (3) an elaborate analysis of generalization capabilities of the IMPU method, and (4) the definition, analysis, and comparison of various transient-based feature types.

The outline of this paper is as follows. In Sect. 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 Sect. 3, different transient-based output features are introduced. In Sect. 4, the introduced technique is applied on a (simulated) rigid double beam model and results are analyzed. Additionally, this section analyzes the generalization capabilities of the aforementioned transient-based features. Then, in Sect. 5, the IMPU method is applied on a roll plane model, and a brief (quantitative) comparison is made with the application of an extended Kalman filter to update parameter values. Finally, conclusions are discussed and recommendations for future work are given in Sect. 6.

## 2 Model updating using inverse mapping models

### 2.1 Problem definition

It is assumed that a DT of a dynamical system is available in the form of a (nonlinear) reference model of which the dynamics may be 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 time derivatives, $\dot{\boldsymbol{q}}$, where $n_{\text{DoF}}$

represents the number of DoFs in the model. Furthermore, $\boldsymbol{y} \in \mathbb{R}^{n_{\text{out}}}$ is a vector containing $n_{\text{out}}$ outputs and $\boldsymbol{u} \in \mathbb{R}^{n_{\text{in}}}$ contains $n_{\text{in}}$ 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 EoMs, e.g., spring and damping constants (e.g., representing mechanical connections), are used. Although in this paper mechanical systems are considered, the presented methodology is, in fact, applicable to general nonlinear dynamic systems and therefore suited for the use in the DT field where models are often multi-disciplinary. Note that the introduced methodology is, in principle, applicable to any (traditional) dynamical model, i.e., models that are not necessarily used within the DT context (in which, among others, control, SHM, and visualization are also incorporated). The benefits of the proposed methodology are, however, especially relevant in the DT context, as discussed in Sect. 1.

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, leading to a DT that may inaccurately represent its physical counterpart. Therefore, we define a unique 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.

*Remark 1* Since, in practice, a correct model structure may prove difficult to achieve, ongoing research by the authors is concerned with (simultaneously) updating parameter values and model structure.

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 discrete time series data $\bar{\mathrm{y}}_i$ for the $i$th output signal $\bar{y}_i(t)$:

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

where $N$ indicates the number of equidistant time points 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 $\mathrm{y}_i(\boldsymbol{p})$, defined similarly to (3). If $\boldsymbol{p} \neq \bar{\boldsymbol{p}}$, there exists a mismatch between $\mathrm{y}_i(\boldsymbol{p})$ and $\bar{\mathrm{y}}_i$, even if the measurement noise $\bar{\boldsymbol{w}}$ 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:

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

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, $\boldsymbol{\psi}_i(\boldsymbol{p}) := \boldsymbol{\psi}_i(\mathrm{y}_i(\boldsymbol{p}))$. Furthermore, $\mathcal{E} : \mathbb{R}^{n_\psi} \to \mathbb{R}$ is a cost functional and $\bar{\boldsymbol{\psi}}$ contains the concatenated measured features of all $n_{\text{out}}$ 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 for a DT application. 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 Sect. 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, inverse mapping models are employed, as proposed in [60]. In contrast to forward, FP 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 $\mathcal{I} : \mathbb{R}^{n_\psi} \rightarrow \mathbb{R}^{n_p}$:

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

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 [66]. Moreover, in the online phase, inferring parameter values using ANNs is computationally efficient, as is also discussed in [59], where ANNs are used as surrogates for FP models. Note that, in contrast to parameter updating methods such as PINNs or filter-based techniques (see Sect. 1), known physics are not explicitly used in the ANN. However, since training data are generated using the FP model, the physics governing this model are implicitly incorporated in and learned by the ANN.

*Remark 2* In the case that the model structure is incorrect, the IMM method will try to find suitable parameter values such that the measured features are mimicked as closely as possible (but inevitably with loss of accuracy) using the model parameterized with the updated parameter values, i.e., the updated model. Note that in this case, the identified parameter values do not fully represent their physical counterparts as intended in the reference model and should, therefore, be handled with care for, e.g., SHM.

Next, Sect. 2.2.1 gives a general introduction to feed-forward ANNs. Afterward, the offline and online phases of the IMPU method are discussed in Sects. 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 $\bar{\boldsymbol{\psi}}$ to a vector containing parameter values $\hat{\boldsymbol{p}}$. The ANN consists of multiple layers consisting of neurons that are interconnected between neighboring layers, see Fig. 1. In the input layer, each entry of the feature vector $\bar{\boldsymbol{\psi}}[j]$, $j \in \{1, \ldots, n_\psi\}$, is assigned its own neuron. In the output layer, the parameter vector $\boldsymbol{p}$ is used likewise. Between the in- and output layer, $n_r$ hidden layers containing $n_z^{(r)}$ neurons in hidden layer $r$ ($r$ is an element of $\{1, \ldots, n_r\}$) are located. Here, $n_r$ and $n_z^{(r)}$ should be chosen large enough to capture the complexity of the inverse mapping[1], but not too large to prevent, e.g., improper training and poor generalizations. It is therefore advisable to optimize these values by employing hyperparameter tuning [69].

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

$$\boldsymbol{z}^{(r)}[j] = \alpha({\boldsymbol{w}_j^{(r)}}^\top \boldsymbol{z}^{(r-1)} + b_j^{(r)}), \tag{8}$$

where $\alpha(\cdot)$ represents an activation function that allows for nonlinearities to be introduced in the mapping. Furthermore, $\boldsymbol{w}_j^{(r)} \in \mathbb{R}^{n_z^{(r-1)}}$ represents a vector containing trainable weights between neuron $j$ in layer $r$ and all neurons in layer $r-1$, and $b_j^{(r)}$ represents a trainable bias that allows for an affine input to $\alpha(\cdot)$. Here, $r = 0$ and $r = n_r + 1$ denote the input and output layer, respectively. Since the bias can also be regarded as a weight, from here on, the term 'weight' refers to both the weights $\boldsymbol{w}$ and the biases $b$. For a more detailed discussion of ANNs, the reader is referred to [66].

#### 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 employed.

As shown schematically in Fig. 2, depicting the offline phase of the IMPU method, the first step is training/validation/testing data generation, where these data types are distinguished by three distinct color-coded lines. Note that, although the testing data are not used in the offline phase, in Sects. 4 and 5, these data are used to test performance of the trained ANN. In Fig. 2, data

[1] Although, theoretically, a single-hidden layer network is able to approximate any function [67], using multiple layers with fewer nodes may decrease training and inference time [68].

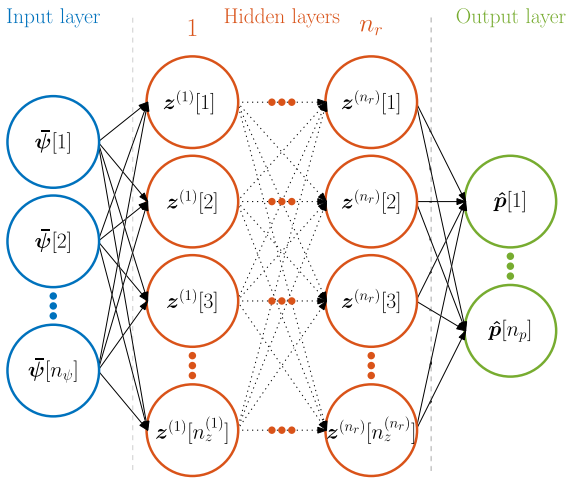


**Fig. 1** Structure of a fully connected feed-forward ANN with $n_r$ hidden layers

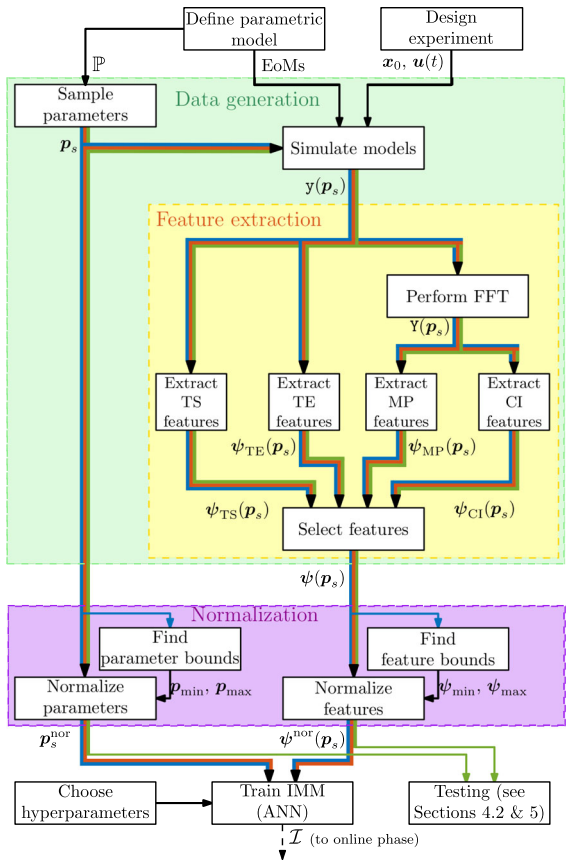


**Fig. 2** Schematic representation of the offline phase of the IMPU method. The blue (—), orange (—), and green (—) lines indicate training, validation, and testing data, respectively. Black lines (—) refer to universal variables

generation (area in light green) is performed for each sample $s$ separately. First, 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 [70]). 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 perform a simulation an experiment, with predefined initial conditions $\boldsymbol{x}_0$ and excitations $\boldsymbol{u}(t)$. This results in transient data $\mathrm{y}(\boldsymbol{p}_s)$, from which a set of features $\boldsymbol{\psi}(\boldsymbol{p}_s)$ is extracted (for which different feature types may be used) using (4). Note that the feature extraction is performed for each output signal individually, after which features extracted from the output signals are concatenated using (6). In Fig. 2, four feature types are extracted (see Sect. 3 for their definitions) from which features are selected[2]. Furthermore, it is remarked that we assume that the true parameter values will lie in $\mathbb{P}$. Therefore, provided that we use sufficient training samples, all (nonlinear) responses in $\mathbb{P}$ are captured in the training data (note that in normalized response features also still a nonlinear dependency of response features and parameters is represented if it is present in the original data). Consequently, encountering such responses in the testing data/online will be recognizable for the trained ANN. Furthermore, note that the lack of hats and bars for $\boldsymbol{p}_s$ and $\boldsymbol{\psi}(\boldsymbol{p}_s)$ indicates that these variables correspond to training data.

After data generation, to increase robustness of the ANN [71], the inputs and outputs of the training data, i.e., $\boldsymbol{\psi}(\boldsymbol{p}_s)$ and $\boldsymbol{p}_s$ for $s \in \{1, \ldots, n_s\}$, are normalized such that, per entry in both $\boldsymbol{\psi}(\boldsymbol{p}_s)$ and $\boldsymbol{p}_s$, the minimum and maximum value across all $n_s$ training samples equal 0 and 1, respectively. For more details about normalization, see Appendix A.

*Remark 3* 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 guarantee a good signal-to-noise ratio), excite relevant nonlinearities, and take the bounds of the physical system into consideration, see, e.g., [31].

[2] Although multiple selection strategies can be applied, in this research, feature selection is limited to selecting a (sub)set of features belonging to a single type only.

Having generated the normalized training data, the weights of the ANN with a user-defined structure (i.e., hyperparameters such as activation functions and number of layers and nodes) are tuned. First, the associated weights are initialized with random values. Subsequently, during iterative application of $n_e$ epochs these weights are tuned, where in one epoch all training data are used once. At the start of each epoch, all training data are randomly divided over $n_b$ batches (subsets of training data containing multiple training samples). Then, for each sample $s$ in a batch, the following sequence of steps is performed:

1. The ANN (parameterized using the current values for the weights) is used to estimate the parameter values given the features of sample $s$, i.e., $\breve{\mathcal{I}}_k\big(\boldsymbol{\psi}(\boldsymbol{p}_s)\big)$, where the breve indicates that the ANN weights are still being tuned.
2. The cost function value is calculated using $\boldsymbol{p}_s$ and its current estimate $\breve{\mathcal{I}}_k\big(\boldsymbol{\psi}(\boldsymbol{p}_s)\big)$. In this paper, the squared error is used as the cost function:

$$\epsilon = \big(\breve{\mathcal{I}}\big(\boldsymbol{\psi}(\boldsymbol{p}_s)\big) - \boldsymbol{p}_s\big)^\top \big(\breve{\mathcal{I}}\big(\boldsymbol{\psi}(\boldsymbol{p}_s)\big) - \boldsymbol{p}_s\big), \tag{9}$$

   Note that, in contrast to (5), (9) minimizes the difference between the estimated and known training parameter values instead of features, due to the inverse nature of the mapping.
3. Using backpropagation, the gradients of $\epsilon$ with respect to all weights are calculated.

Using the gradients obtained in step 3 for all samples in the batch, ANN weights are updated such that the mean squared error (MSE) in the parameter values over all samples in the batch is minimized. After this update, steps 1–3 are repeated for a new batch, until all batches have been processed. Then, the validation data are used to calculate the validation loss (MSE in the parameter values). Here, the validation data are obtained in an equivalent manner as the training data (with distinctively sampled parameter values), see Fig. 2, and used to evaluate performance of the ANN without being biased towards the training data. If the validation loss has not decreased for $n_{\text{es}}$ consecutive epochs or all $n_e$ epochs have been processed, training is stopped and the ANN with the lowest validation loss is saved; otherwise, a new epoch is started. This is referred to as patience-based early stopping and is employed to prevent overfitting [72]. For more information about backpropagation and ANN training in general, the reader is referred to [66].

### 2.2.3 Online phase

After having trained the ANN offline with simulated data, leading to the inverse trained mapping in (7), in the online phase, 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 Fig. 3, this is achieved by feeding normalized output features $\bar{\boldsymbol{\psi}}^{\text{nor}}$ to the ANN see (7). Here, $\bar{\boldsymbol{\psi}}^{\text{nor}}$ is obtained by normalizing (similar to normalization of the training data) the features extracted from a measurement of a physical system. Recall that an overbar is used to indicate that these features relate to the physical system. Since the ANN has been trained using normalized features and parameter values to increase robustness, in the online phase, again normalized features and parameter values are utilized. To maintain consistency, normalization in the online phase is performed using bounds $\boldsymbol{\psi}_{\min}$ and $\boldsymbol{\psi}_{\max}$ as obtained during training, see Appendix A. The inferred (or updated) normalized physical parameter values are then denormalized and substituted in the parameterized reference model (1) resulting in the updated model such that the DT better mimics the true system. Furthermore, although the model constituting a DT should in general be computationally efficient (e.g., for control purposes), the IMPU method does not require online evaluation of the parameterized model (1), since in the online phase only measurements are used and the ANN is evaluated for the measured features to generate a parameter estimate. Defining parametric models with low online model evaluation time, i.e., real-time simulation, which can be achieved using, e.g., parametric model order reduction techniques is therefore regarded as out of scope for this paper.

Due to the use of transient data to accommodate for nonlinear systems, in contrast to (modal) characteristics of linear systems as previously used in [60,62,63], 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 and known in advance. Consequently, this updating method is only suited for applications with known forcing which is relevant, e.g., in high-performance motion stage systems with known reference trajectory inputs.

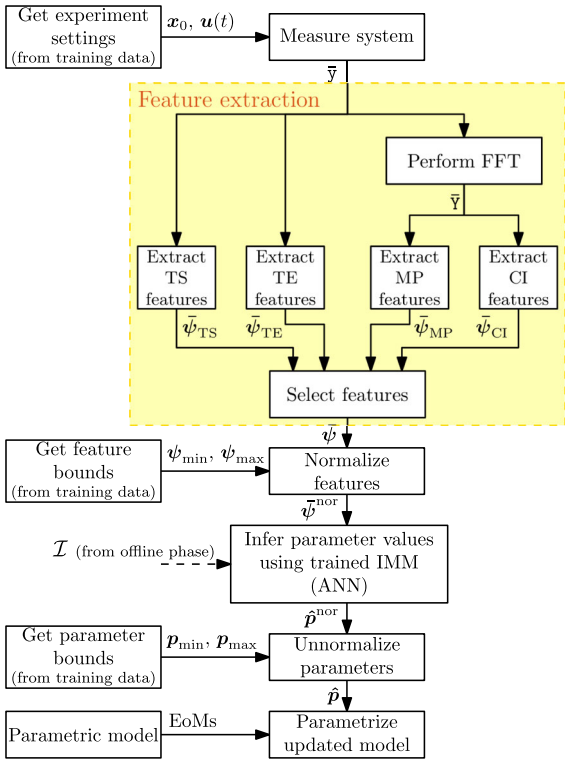


**Fig. 3** Schematic representation of the online phase of the IMPU method

*Remark 4* 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 definitions for the (not yet normalized) feature vector $\boldsymbol{\psi}_i(\boldsymbol{p})$, containing a set of $n_\psi$ (depending on the choice of the user) features that are used as inputs to the ANN, are given. Here, these features are extracted from sampled transient data $\mathtt{Y}_i(\boldsymbol{p})$ as obtained from a simulated (for a model parameterized with $\boldsymbol{p}$) or measured experiment.

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 value variations. 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). For the validation/test and online measured data, the discussed feature extraction methodologies are applied identically.

The introduced feature types are divided in time- and frequency-domain features, which will be discussed in the coming Sects. 3.1 and 3.2, respectively.

### 3.1 Time-domain features

Here, feature types are introduced that directly use the sampled transient data $\mathtt{Y}$. In Figs. 2 and 3, these feature types are also indicated in the two left 'extract features' blocks in the feature extraction part (marked in yellow).

*Time samples*

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

$$\boldsymbol{\psi}_{\mathrm{TS}} = \mathtt{Y}[\boldsymbol{v}_{\mathrm{TS}}], \tag{10}$$

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 use the magnitude at and temporal location of a selection of $n_{\mathrm{TE}}$ local extrema in the time domain signal:

$$\boldsymbol{\psi}_{\mathrm{TE}} = \left[\mathtt{Y}[\boldsymbol{v}_{\mathrm{TE}}]^\top, \mathtt{t}[\boldsymbol{v}_{\mathrm{TE}}]^\top\right]^\top. \tag{11}$$

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 $n_{\mathrm{TE}}$ 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 value beyond the peak itself [73]). As, for different samples obtained for distinct parameter values sets, the measured signals may not have the same number of extrema, $n_{\mathrm{TE}}$ 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. For example, the heights of consecutive peaks in an impulse response (which decrease in magnitude) are indicative about the amount of damping in a system. Consequently, only few of these features can already contain a lot of information about the transient response that is closely related to physical parameters such as mass, damping, and stiffness parameters. Therefore, compared to, e.g., TS features, the ANN only needs to learn a relatively simple IMM, potentially boosting its accuracy and simultaneously lowering training and inference time due to the smaller set of features. A disadvantage, however, is that the temporal location of an extremum is relatively sensitive to noise, potentially causing the true location of the extremum to be identified at one of the neighboring 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

Alternatively, features can be 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, as indicated in Figs. 2 and 3, first, the measured/simulated time domain data $\mathtt{y}$ are transformed to the frequency domain:

$$\mathtt{Y}(f_j) = \mathcal{F}(\mathtt{y}(t_k)), \tag{12}$$

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) algorithm is used.

Since the discrete frequency domain data are represented in a set of complex numbers, there are two options to pass this data to the ANN (which requires real valued data): 1) using the Magnitude and Phase information (MP): see (13), or 2) using the Real and Imaginary components of the frequency domain data (RI): see (14):

$$\boldsymbol{\psi}_{\mathrm{MP}} = \left[|\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]|^\top, \angle\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]^\top\right]^\top, \tag{13}$$

$$\boldsymbol{\psi}_{\mathrm{RI}} = \left[\Re\left(\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]\right)^\top, \Im\left(\mathtt{Y}[\boldsymbol{v}_{\mathrm{FT}}]\right)^\top\right]^\top, \tag{14}$$

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 and are therefore excluded from 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. For example, at bins corresponding to high frequencies, the signals may be dominated by noise. Excluding these bins from $\boldsymbol{v}_{\mathrm{FT}}$ can improve the inverse mapping learned by the ANN. In this paper, FT-based features that are limited to the information in some FBoIs are indicated by the suffix (FBoI).

## 4 Illustrative case study on double beam system

In this section, the inverse mapping parameter updating method and various feature types are applied on a case study of a nonlinear multibody rigid double beam (RDB) 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 Sect. 4.1. Additionally, this section discusses the employed ANN and accompanying (training) settings. In Sect. 4.2, results are analyzed in terms of accuracy of the inferred parameter values and responses of the updated model/DT. Finally, an extensive analysis of the generalization capabilities of the ANN is given in Sect. 4.3.

### 4.1 Problem setting

#### *4.1.1 Model description*

Figure 4 shows the side view of the multibody system academic demonstrator by means of which the introduced methodology is demonstrated in this section. 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]. The location of the 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: $\boldsymbol{\rho}_{\mathrm{CoM}} = \left(l_{\mathrm{CoM},1}\cos(y_2) - l_{\mathrm{CoM},2}\sin(y_2)\right)\boldsymbol{e}_1 + \left(l_{\mathrm{CoM},1}\sin(y_2) + l_{\mathrm{CoM},2}\cos(y_2)\right)\boldsymbol{e}_2$, where $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ represent unit vectors of a (global) fixed reference frame as indicated in Fig. 4.

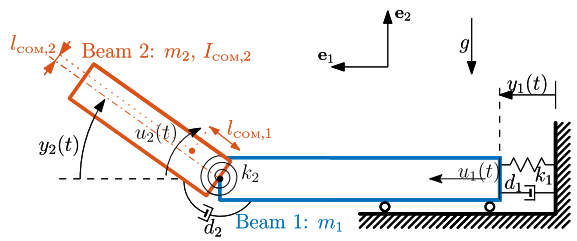


**Fig. 4** Schematic representation of the RDB academic demonstrator model

**Table 1** Updating parameter space bounds for the RDB model

| Parameter | Lower bound | Upper bound | Unit |
|---|---|---|---|
| $d_1$ | 0.8 | 1.2 | $\frac{\mathrm{N\cdot s}}{\mathrm{m}}$ |
| $d_2$ | $1.75 \times 10^{-4}$ | $2.25 \times 10^{-4}$ | $\frac{\mathrm{N\cdot m\cdot s}}{\mathrm{rad}}$ |
| $k_1$ | 5 | 15 | $\frac{\mathrm{N}}{\mathrm{m}}$ |
| $k_2$ | $2.7 \times 10^{-2}$ | $4.5 \times 10^{-2}$ | $\frac{\mathrm{N\cdot m}}{\mathrm{rad}}$ |

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 $\theta_{k_2}$, 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.1.

In the following case study, the values of parameters $d_1$, $d_2$, $k_1$, and $k_2$ of the model/DT 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:

$$u_1(t) = \begin{cases} 5\ \mathrm{N} & \text{if } 0.2 \le t < 0.25, \\ 0\ \mathrm{N} & \text{else}, \end{cases}$$
$$u_2(t) = \begin{cases} 0.075\ \mathrm{Nm} & \text{if } 0.2 \le t < 0.25, \\ 0\ \mathrm{Nm} & \text{else}. \end{cases} \tag{15}$$

Here impulse-like excitations are used since these yield free vibration responses that vary significantly when

**Table 2** Constant parameter values of RDB model

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

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., [74,75].

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[3], 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, the length of the simulated experiment is chosen long enough such that sufficient, relevant dynamics is captured, but not too long to prevent excessive data generation times[4]. Furthermore, $N$ is chosen as a power of 2 to make the Fast Fourier Transform (FFT) algorithm as efficient as possible. Note that, in practice, the simulated signal may require down-sampling such that it matches the measurement sampling rate. 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 [76], the addition of noise to training data improves robustness of the ANN. Examples of possible responses using the above experiment design are provided in Sect. 4.1.2.

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' behavior of the system is presented here. Each model, parameterized using one of the parameter cases, is linearized around its static equilibrium point. The resulting eigenvalues (with positive imaginary part) $\lambda_i$, (damped) eigenfrequencies $f_{i,\text{eig}}$, and their corresponding, modal, damping coefficients $\xi_i$, 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 Sect. 4.1.1. Furthermore, features are extracted, conform the theory in Sect. 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 `mpp(y)` is used for output signal `y` (as used in the timepeaks Matlab function [73]) and is defined as

$$\texttt{mpp(y)} = 0.15\,(\max(\texttt{y}) - \min(\texttt{y}))\,, \tag{16}$$

where `y` denotes a sampled output signal. As a result, a peak is only defined if the value of `y` between that peak and the closest peak with a higher magnitude drops by a minimal amount of `mpp` [73]. Note that the definition of `mpp(y)` should be tailored towards the encountered

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

[4] In practice, the length of an experiment also depends on the duration of operational trajectories and/or the time available to perform a dedicated updating experiment during operation.

**Table 3** Different parameter cases with corresponding updating parameter values, resulting eigenvalues, (damped) eigenfrequencies, and modal damping coefficients of the linearized RDB models. Note that, $\xi_i = \frac{-\Re(\lambda_i)}{2\pi f_{i,\text{eig}}}$ in principle is only 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 | $d_1$ [$\frac{\text{N·s}}{\text{m}}$] | $d_2$ [$\frac{\text{N·m·s}}{\text{rad}}$] | $k_1$ [$\frac{\text{N}}{\text{m}}$] | $k_2$ [$\frac{\text{N·m}}{\text{rad}}$] | $\lambda_1$ [rad/s] | $\lambda_2$ [rad/s] | $f_{1,\text{eig}}$ [Hz] | $f_{2,\text{eig}}$ [Hz] | $\xi_1$ [-] | $\xi_2$ [-] |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.0 | $2.00 \times 10^{-4}$ | 10 | $3.6 \times 10^{-2}$ | $-0.217 + 2.074j$ | $-0.331 + 10.477j$ | 0.330 | 1.668 | 0.104 | 0.032 |
| 2 | 0.8 | $1.75 \times 10^{-4}$ | 5 | $2.7 \times 10^{-2}$ | $-0.174 + 1.464j$ | $-0.290 + 9.034j$ | 0.233 | 1.438 | 0.118 | 0.032 |
| 3 | 0.8 | $1.75 \times 10^{-4}$ | 15 | $4.5 \times 10^{-2}$ | $-0.174 + 2.548j$ | $-0.290 + 11.776j$ | 0.406 | 1.874 | 0.068 | 0.025 |
| 4 | 1.2 | $2.25 \times 10^{-4}$ | 5 | $2.7 \times 10^{-2}$ | $-0.261 + 1.451j$ | $-0.372 + 9.031i$ | 0.231 | 1.437 | 0.177 | 0.041 |
| 5 | 1.2 | $2.25 \times 10^{-4}$ | 15 | $4.5 \times 10^{-2}$ | $-0.261 + 2.540j$ | $-0.372 + 11.774j$ | 0.404 | 1.874 | 0.102 | 0.032 |

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 Figs. 5 and 6, 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 Figs. 5 and 6, respectively. Furthermore, Figs. 7 and 8 show the corresponding Fourier transforms of the TS (dense) response features using the MP representation, and the RI representation, respectively. Additionally, Figs. 7 and 8 indicate the features corresponding to the FBoIs using color-coded squares.

#### *4.1.3 Artificial neural network*

As mentioned in Sect. 2.2, the IMM is constituted by a (trained) feed-forward ANN, see Fig. 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

$$\alpha(a) = \begin{cases} a \text{ if } a > 0, \\ 0 \text{ else}, \end{cases} \tag{17}$$

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 [77].

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 [78], where the mean squared error is minimized over $n_e = 200$ epochs in 20 batches of $n_b = 50$ samples. Additionally, patience-based early stopping is included, meaning that training is stopped if the validation loss, calculated for 1000 validation samples, has not decreased in $n_{\text{es}} = 40$ 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. Since ANN hyperparameter tuning is not trivial, multiple ANNs with different combinations of empirically chosen hyperparameters have been trained and evaluated on the test data. The ANN presented here was chosen due to a favorable combination of parameter estimation accuracy and complexity (number of neurons and layers, and training time). This being said, considering the large hyperparameter space, these hyperparameters are not expected to be optimal. Therefore, optimization of the ANN hyperparameters is part of current research by the authors.

Using a single core 2.60 GHz CPU, the offline generation of the training data, i.e., simulating the reference model for all $n_s = 1000$ training samples, takes

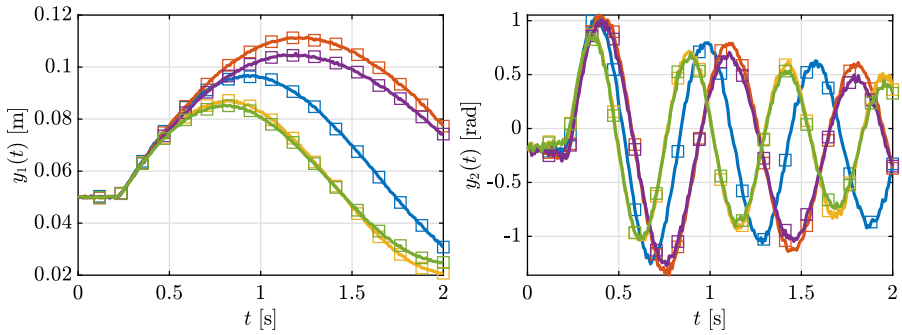


**Fig. 5** Time domain simulation responses of the RDB model with temporally equidistant TS (dense) features (consisting of all points used to draw each response). Additionally, the TS (sparse) features are indicated by the (colored) squares (□). The (line) colors correspond to parameter cases 1 (——),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

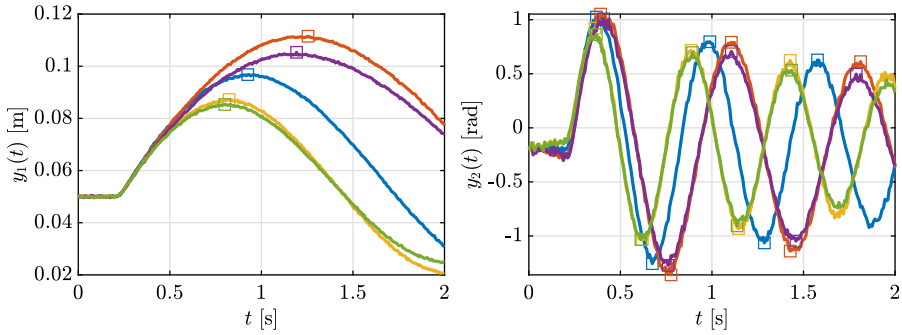


**Fig. 6** Time domain simulation responses of the RDB model with TE features (local extrema) indicated by the (colored) squares (□). Note that the TE feature vector contains pairs of the temporal location and height of each extremum. The line colors correspond to parameter cases 1 (——), 2 (——), 3 (——), 4 (——), and 5 (——)

approximately 3 min. 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 of the DT.

## 4.2 Inference results: parameter updating using the ANN

In the following, the accuracy of the inferred parameter values and updated model/DT is analyzed in Sects. 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. These test samples are obtained similarly to the training and validation samples, see Fig. 2. 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, $\boldsymbol{\varepsilon}_{\bar{s}} \in$

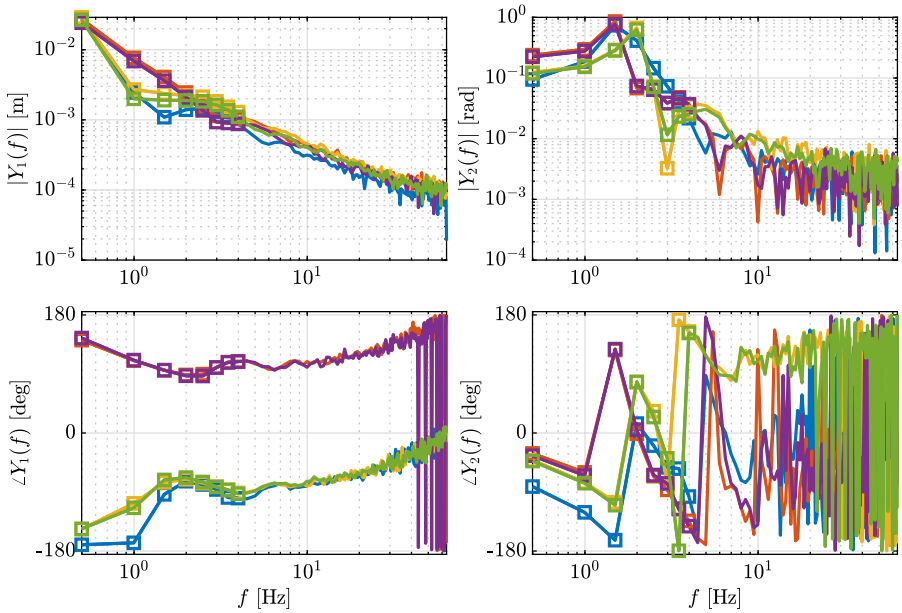


**Fig. 7** Fourier transform of TS (dense) simulation responses of the RDB model represented by magnitude and phase. The MP features consist of all points used to draw each curve. Furthermore, the MP (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—)

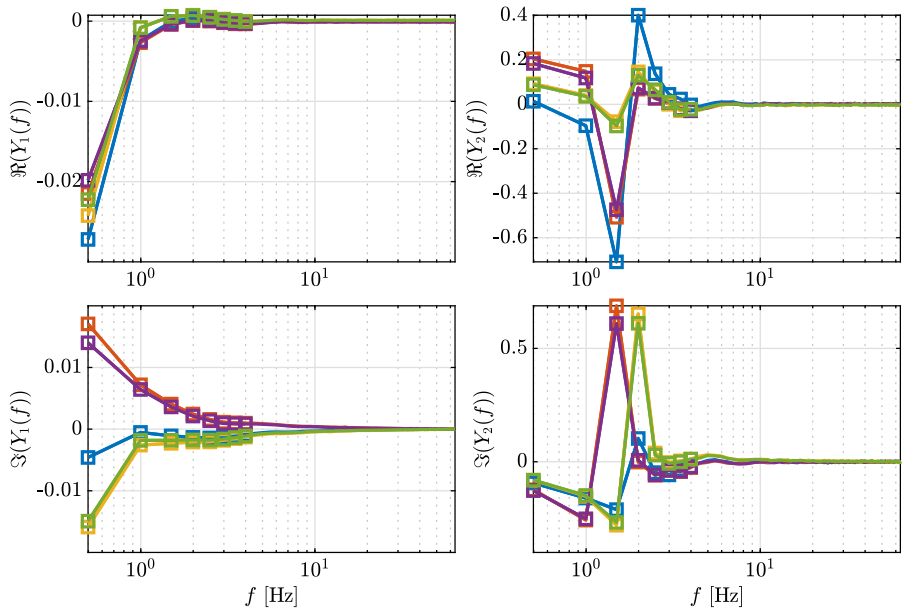


**Fig. 8** Fourier transform of TS (dense) simulation responses of the RDB model represented using real and imaginary values. The RI features consist of all points used to draw each curve. Furthermore, the RI (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter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

$\mathbb{R}^{n_p}$, of the parameter estimate $\hat{\boldsymbol{p}}_{\bar{s}}$:

$$\boldsymbol{\varepsilon}_{\bar{s}} = \left(\hat{\boldsymbol{p}}_{\bar{s}} - \bar{\boldsymbol{p}}_{\bar{s}}\right) \oslash \left(\bar{\boldsymbol{p}}_{\bar{s}}\right), \tag{18}$$

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 9 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

**Table 4** ANN structure for RDB use case

| Layer | Number of neurons | Activation function |
|---|---|---|
| Input | $n_\psi$ | – |
| Hidden 1 | 200 | ReLU |
| Hidden 2 | 100 | ReLU |
| Output | $n_p(=4)$ | Linear |

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 Sect. 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 Figs. 5 and 8. 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 Fig. 10, 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 [79]) standard deviation, $\boldsymbol{\sigma}_\varepsilon$, and the mean absolute relative error, $\boldsymbol{\mu}_{|\varepsilon|}$:

$$\boldsymbol{\mu}_\varepsilon = \frac{1}{n_{\bar{s}}} \sum_{\bar{s}=1}^{n_{\bar{s}}} \boldsymbol{\varepsilon}_{\bar{s}}, \tag{19}$$

$$\boldsymbol{\sigma}_\varepsilon = \sqrt{\frac{1}{n_{\bar{s}}-1} \sum_{\bar{s}=1}^{n_{\bar{s}}} \left(\boldsymbol{\varepsilon}_{\bar{s}} - \boldsymbol{\mu}_\varepsilon\right) \otimes \left(\boldsymbol{\varepsilon}_{\bar{s}} - \boldsymbol{\mu}_\varepsilon\right)}, \tag{20}$$

$$\boldsymbol{\mu}_{|\varepsilon|} = \frac{1}{n_{\bar{s}}} \sum_{\bar{s}=1}^{n_{\bar{s}}} |\boldsymbol{\varepsilon}_{\bar{s}}|, \tag{21}$$

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 Sect. 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 Fig. 5. Secondly, the results obtained using time extrema features are comparatively inaccurate for $d_1$ and $k_1$, which can be ascribed to a low number of features[5] and a relatively large influence of noise. As there are more TE features for signal $y_2(t)$, we see that $d_2$ and $k_2$ are estimated with reasonable accuracy despite the low number of features due to the smart choice of features.

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 Sect. 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 Figs. 11 and 12, the MP

[5] Since $d_1$ and $k_1$ dominantly influence $y_1(t)$, there are only two TE features (one extremum) that can be used to determine these updating parameters, see Fig. 6.

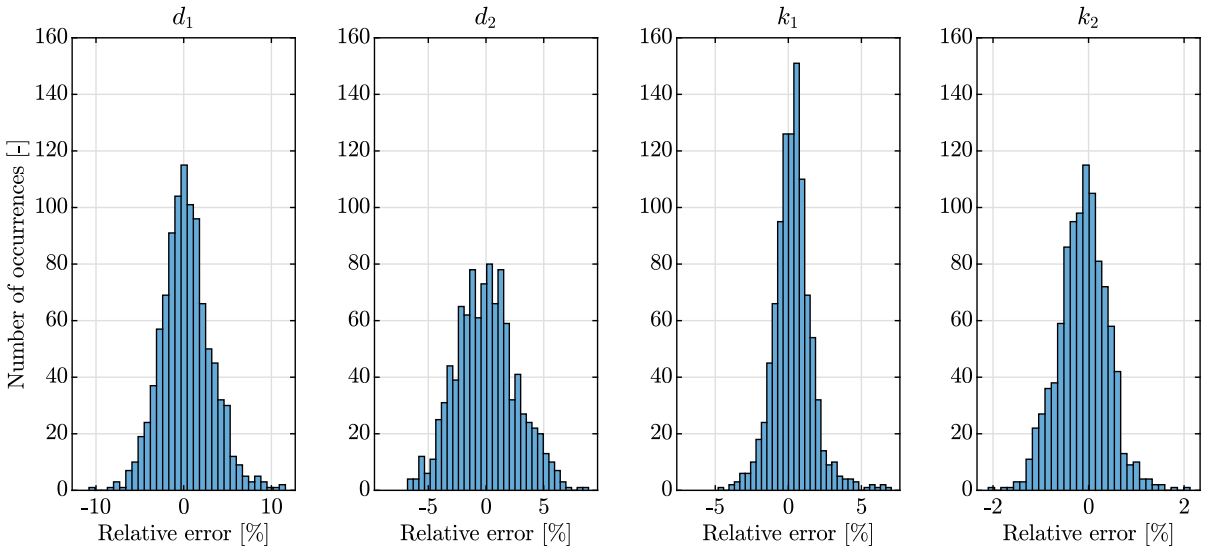


**Fig. 9** Distribution, per updating parameter, of relative inferred parameter errors, $\boldsymbol{\varepsilon}_{\bar{s}}$, as obtained using the RI (FBoI) output features and $n_s = 1000$ training samples in the RDB use case

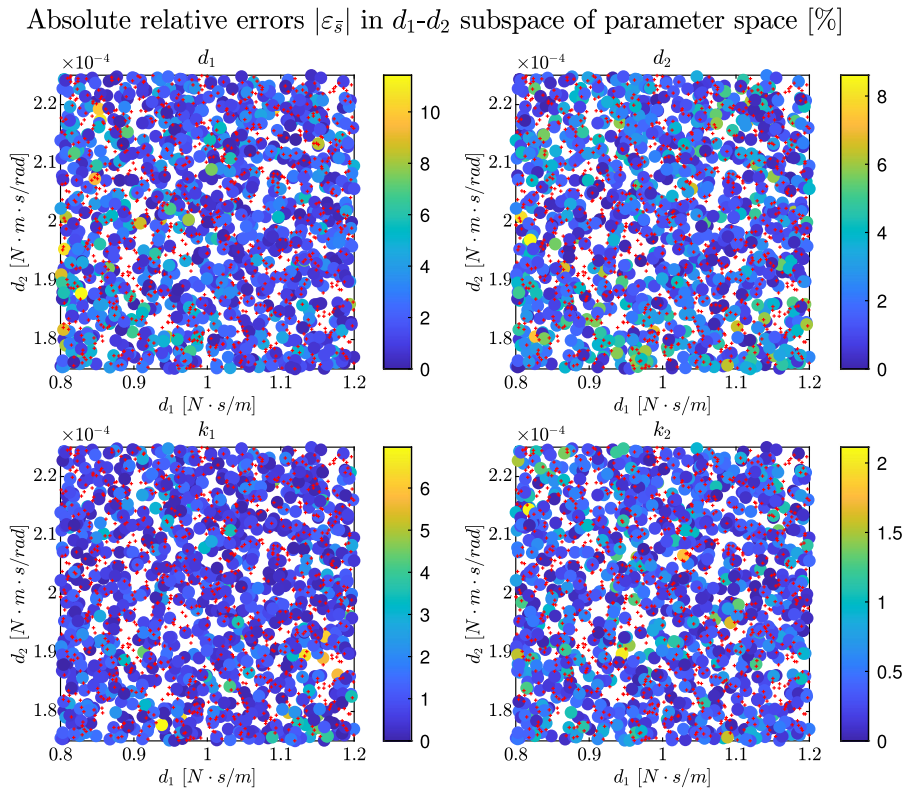


**Fig. 10** Absolute relative error $|\varepsilon_{\bar{s}}|$ of each test sample $\bar{s}$ in the $d_1$-$d_2$ subspace of $\mathbb{P}$ per updating parameter in the RDB use case, as obtained by using the RI (FBoI) output features with $n_s = 1000$. Red plus signs ( + ) indicate locations of training data samples

**Table 5** Relative error metrics per updating parameter for all feature types in the RDB use case, for $n_s = 1000$. Furthermore, the number of features per output feature type, $n_\psi$, is listed

| Feature Type | $n_\psi$ | $\boldsymbol{\mu}_\varepsilon$ [%] | | | | $\boldsymbol{\sigma}_\varepsilon$ [%] | | | | $\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ [%] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $d_1$ | $d_2$ | $k_1$ | $k_2$ | $d_1$ | $d_2$ | $k_1$ | $k_2$ | $d_1$ | $d_2$ | $k_1$ | $k_2$ |
| 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 |

(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 Fig. 11, 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 Fig. 12. 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 [80].

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.

### *4.2.2 Updated model accuracy*

As shown in the previous section, the IMPU method is able to infer parameter values with mean absolute relative errors of approximately 2% or lower. However, one of the main purposes of parameter updating is to obtain a more accurate model, such that the Digital Twin accurately represents the true physical system. 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

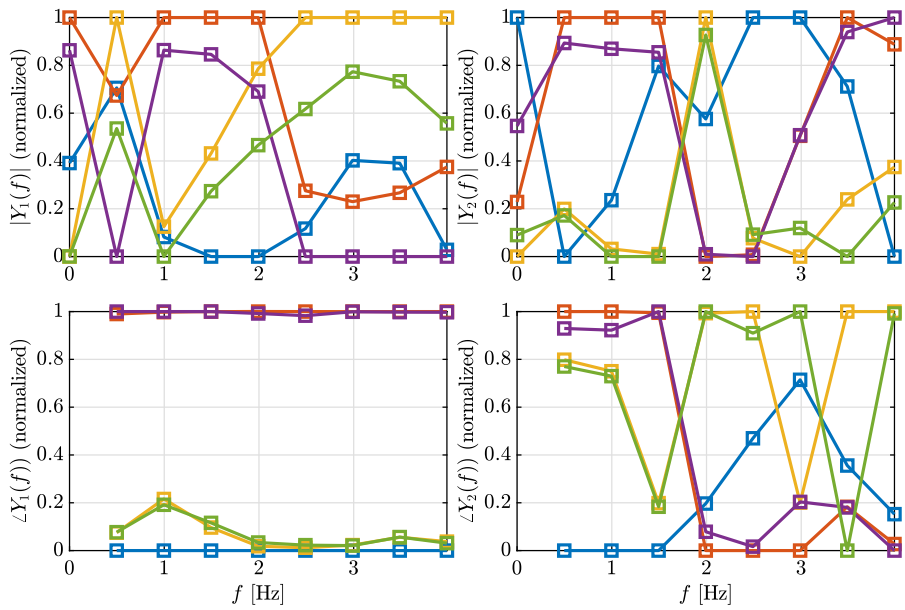


**Fig. 11** MP (FBoI) features, normalized for the five parameter cases, as obtained from the Fourier transform of TS (dense) simulation responses of the RDB model represented using magnitude and phase, zoomed in on the FBoIs. The MP (FBoI) features are indicated by the (colored) squares (□). The line colors correspond to parameter cases 1 (—), 2 (—), 3 (—), 4 (—), and 5 (—)

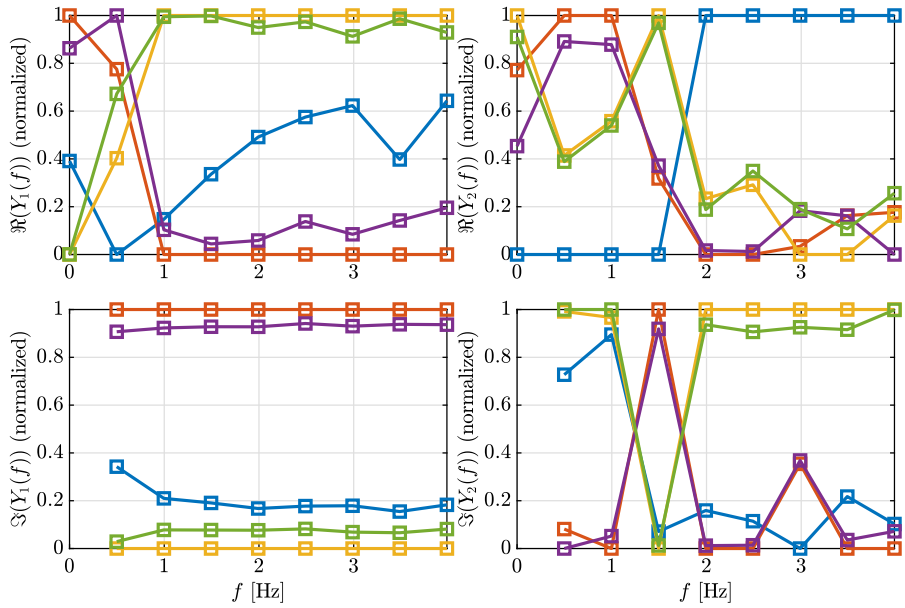


**Fig. 12** RI (FBoI) features, normalized for the five parameter cases, as obtained from the Fourier transform of TS (dense) simulation responses of the RDB model represented using real and imaginary values, zoomed in on the FBoI. The RI (FBoI) features are indicated by the (colored) squares (□). The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

results in the solid blue line in Fig. 13. For this simulation, we have used the same (simulated) experiment settings as discussed in Sect. 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 updated DT, i.e., the reference model parameterized with these updated parameter values, yields the dashed orange line in Fig. 13. 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, Fig. 14 shows the response of the, among others, updated model

**Table 6** Parameter values used to parameterize RDB models used in Figs. 13 and 14

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

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 $\boldsymbol{p} = \boldsymbol{p}_c$. This shows that a Digital Twin that is updated using some updating experiment can be employed to simulate operating scenarios that have not been used for the updating procedure.

## 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 Sect. 4.2.1, which is similar to the analysis in [18], 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 Fig. 15, 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 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 accu-

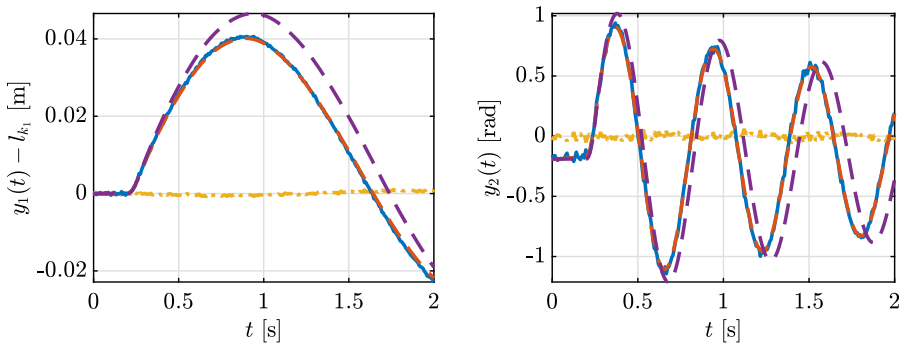


**Fig. 13** Comparison of simulated measurement ( ——— ) and simulation of updated RDB model ( - - - ), for the (simulated) experiment used to train the ANN, i.e., impulse-like excitation and the static equilibrium of the reference system with $\boldsymbol{p} = \boldsymbol{p}_c$ as initial conditions. Furthermore, the error of the updated model ( -·- ) is plotted and, for reference, the simulation is performed using the reference model with $\boldsymbol{p} = \boldsymbol{p}_c$ ( - - - )

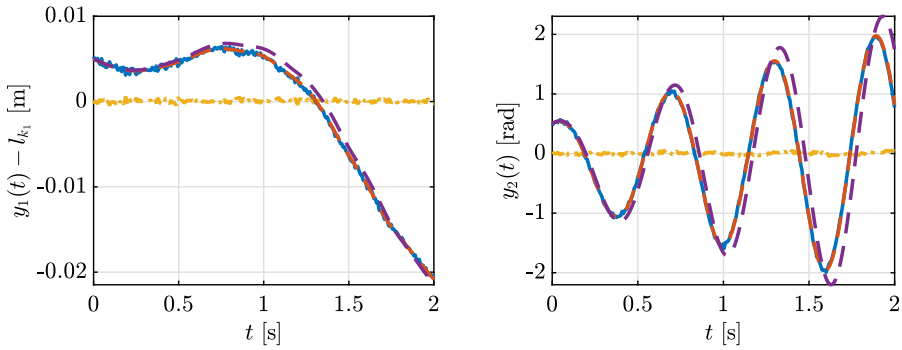


**Fig. 14** Comparison of simulated measurement ( ——— ) and simulation of updated RDB model ( - - - ), for different (simulated) experiment (harmonic excitation and deviating initial conditions) than used to update the model. Furthermore, the error of the updated model ( -·-·- ) is plotted and, for reference, the simulation is performed using the reference model with $\boldsymbol{p} = \boldsymbol{p}_c$ ( - - - )

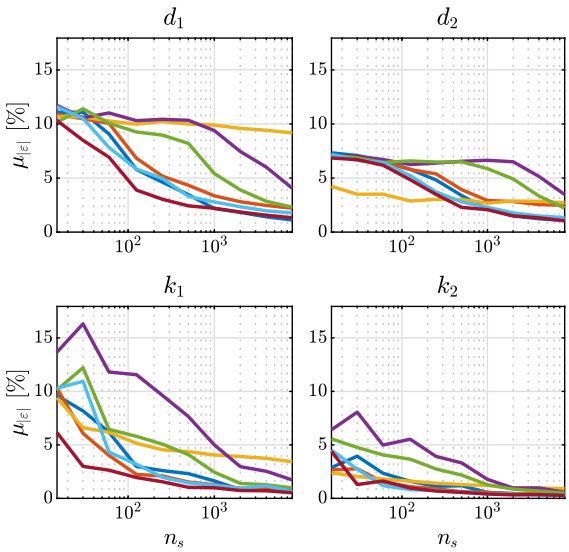


**Fig. 15** Mean absolute relative error as a function of the number of training samples, $n_s$, per updating parameter in the RDB use case, for the different output feature types: TS (dense) ( ——— ), TS (sparse) ( ——— ), TE ( ——— ), MP ( ——— ), RI ( ——— ), MP (FBoI) ( ——— ), and RI (FBoI) ( ——— )

racy 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 TE features is relatively consistent, although comparatively low for some parameters, when varying $n_s$ (especially for $d_1$ and $d_2$). This agrees with the fact that the difference in magnitude between subsequent maxima/minima is closely related to the damping values. This thus shows that the IMM between the damping parameter values and the TE features is relatively easy to learn for the ANN as was hypothesized in Sect. 3.1, i.e., TE features generalize well for different values of $n_s$.

### *4.3.2 Excluded training zones*

In practice, the true parameters of the DT may lie (far) outside the expected parameter space, e.g., when the system is (suddenly) badly damaged. In this case, the IMPU method should recognize this such that the digital twin can be updated for the benefit of SHM.

*Remark 5* The IMPU method can theoretically detect both gradual and sudden degradation/changes to the true parameter values. In case of gradual degradation/changes, SHM using the updated DT can be employed to plan timely maintenance. However, sudden degradation/changes might already damage the system before this change has been detected by the (inverse mapping) parameter updating method. This will, however, be a problem for any model updating strategy.

Therefore, in this section, generalization is analyzed by deliberately omitting parts of the training space and testing how accurate test samples are in distinct 'test-zones.' In the following, changes are only made to the subspace of $\mathbb{P}$ spanned by $d_1$ and $d_2$. Specifically, for

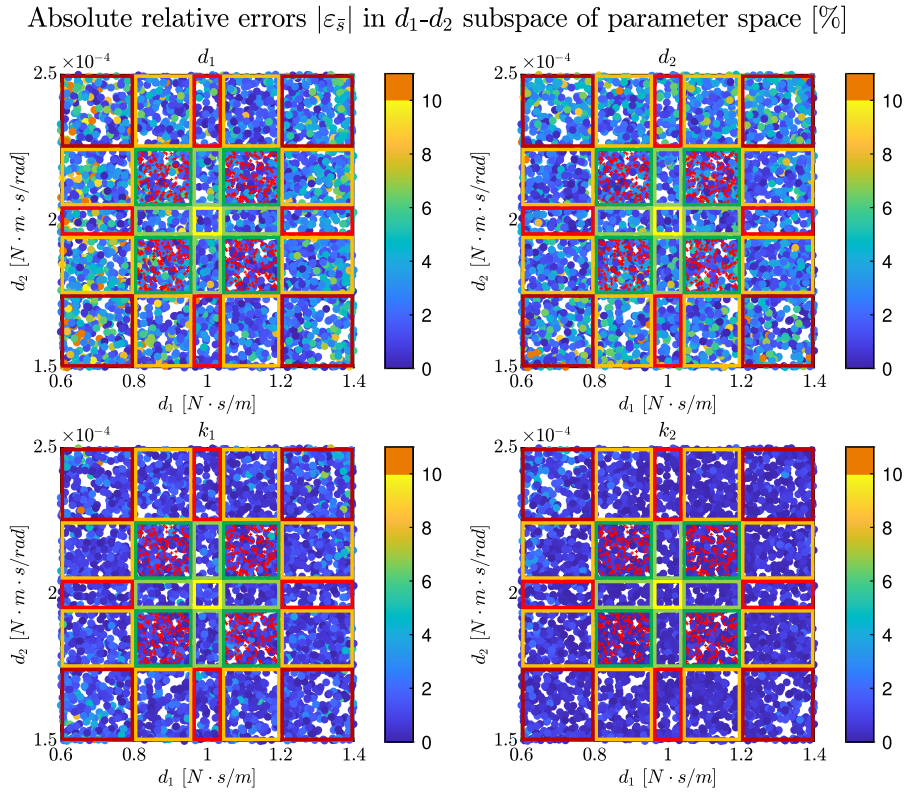


**Fig. 16** Absolute relative error (in %) of each test sample $|\varepsilon|$ (see (18) for definition of $\varepsilon$) in the $d_1$-$d_2$ subspace of $\mathbb{P}$ per updating parameter in the RDB use case, as obtained by using the RI (FBoI) output features with $n_s = 645$. Red plus signs (+) indicate 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

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 Fig. 16, 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 Sect. 4 are reused, of which the samples located in testzones 2 and 3 are excluded. Consequently, as we started with 1000 training samples (as used for Fig. 16), 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 Sect. 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 Fig. 16. In the upper left plot of Fig. 16, 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 Sect. 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 Fig. 17. This is explained by the fact that alterations in training and testing data are restricted to the $d_1$-$d_2$ subspace.

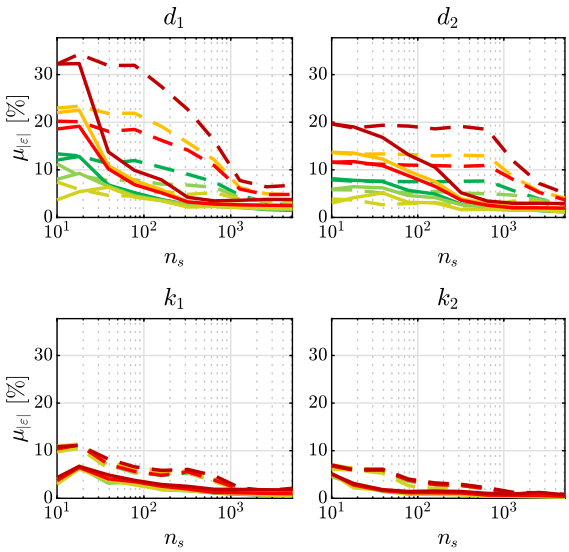


**Fig. 17** Mean absolute relative error (in %) per testzone as a function of the number of training samples, $n_s$, per updating parameter in the RDB use case. The results indicated with solid lines (——) are obtained using RI (FBoI) features, and results indicated with dashed lines (- - -) are obtained using RI features. The colors of the solid/dashed lines correspond to testzones 1 (——), 2 (——), 3 (——), 4 (——), 5 (——), and 6 (——), respectively

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

| Feature Type | $\boldsymbol{\mu}_{\|\varepsilon\|}$ [%] | | | |
|---|---|---|---|---|
| | $d_1$ | $d_2$ | $k_1$ | $k_2$ |
| 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 (21), 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 Fig. 17, mean absolute relative errors $\boldsymbol{\mu}_{|\varepsilon|}$ for the different testzones (as calculated using (21) 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). This observation is, again, closely connected to an aforementioned observation in Sect. 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, Fig. 17 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.

**Table 8** Mean absolute relative error of $d_1$ estimate for testzones 1, 3, and 6, obtained using $n_s = 2569$ training samples in the RDB use case

| Feature Type | $\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ 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 in the RDB use case

| Feature Type | $\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ 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 |

To conclude, in general, if sufficient training data are 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 $\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ in testzone 6 are, with some exceptions for the TE and MP features, approximately two times higher than the values of $\boldsymbol{\mu}_{\lvert\varepsilon\rvert}$ 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. Nevertheless, as estimates will worsen if their true values are farther from the training space, it is advised to treat parameter estimates outside the training space with caution. Incorrect parameter values that are regarded as true, lead to an inaccurate DT which may lead to faulty decisions. For this reason, explicitly quantifying the uncertainty in the inferred parameter estimates is part of ongoing research of the authors. Since the ANN does extrapolate parameter values, one can, however, often recognize that the true system lies outside the training space.

## 5 Case study on roll plane model

In this section, the IMPU method is applied on a different use case. The analyzed model is a four-DoF Roll Plane Model (RPM) of a vehicle with nonlinear suspension driving over a speed bump with constant velocity, as considered in [37], see Fig. 18. The EoMs for this

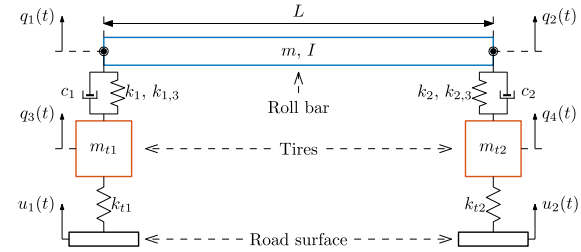


**Fig. 18** Schematic representation of the RPM

model, with DoFs $\boldsymbol{x} = \left[x_1,\ x_2,\ x_3,\ x_4\right]^\top$, are given in Appendix B.2. The non-updating (i.e., known) parameter values are listed in Table 10. The two parameters considered for updating are $k_{t1}$ and $k_{t2}$ (representing the stiffnesses of the tires), for both of which $\mathbb{P}$ ranges from 80 to 120% of their nominal values of 96319.76 N/m.

As outputs, the relative displacement and relative velocity between the roll bar and both tires, i.e., $y_1 = x_1 - x_3$, $y_2 = x_2 - x_4$, $y_3 = \dot{x}_1 - \dot{x}_3$, and $y_4 = \dot{x}_2 - \dot{x}_4$, are measured every 0.3 s for a duration of 3 s. To simulate a real measurement, zero-mean Gaussian noise with a standard deviation of 1% of the value of $y_i(t_k)$[6] is added to $y_i(t_k)$. As inputs, prescribed displacements of the road surface profile $u_1(t)$ and $u_2(t)$ are used, see Appendix B.2. Note that all these settings are equivalent to those in [37].

An ANN (with structure listed in Table 11) is trained for various numbers of training samples and 1000 validation samples. As features, TS, MP, and RI features

[6] Note that, although in [37] (erroneously) a variance of 1% is mentioned, in reality a standard deviation of 1% is used, as is evident from [81] where the same model is used.

**Table 10** Constant parameter values of RPM model

| Parameter | Value | Unit |
|---|---|---|
| $m$ | 580 | kg |
| $m_{t1}, m_{t2}$ | 36.26 | kg |
| $I$ | 63.3316 | $\mathrm{kg \cdot m^2}$ |
| $L$ | 1.524 | m |
| $c_1, c_2$ | 710.70 | $\frac{\mathrm{N \cdot s}}{\mathrm{m}}$ |
| $k_1, k_2$ | 19357.2 | $\frac{\mathrm{N}}{\mathrm{m}}$ |
| $k_{1,3}, k_{2,3}$ | 15000 | $\frac{\mathrm{N}}{\mathrm{m}^3}$ |

**Table 11** ANN structure for RPM use case

| Layer | Number of neurons | Activation function |
|---|---|---|
| Input | $n_\psi (= 40)$ | – |
| Hidden 1 | 100 | ReLU |
| Hidden 2 | 80 | ReLU |
| Hidden 3 | 40 | ReLU |
| Output | $n_p (= 2)$ | Linear |

that used all 10 measured time samples (measurements at $t = 0$ are excluded) for all 4 outputs are used. Evaluating $\boldsymbol{\mu}_{|\varepsilon|}$ of the estimated parameter values for $n_{\bar{s}} = 1000$ testing samples results in the three color-coded solid lines in Fig. 19, where we see that a larger $n_s$ leads to a smaller $\boldsymbol{\mu}_{|\varepsilon|}$. Furthermore, the different feature types show comparable results. Note that, in this case, there is no point in defining FBoIs due to the low number of time samples and high frequency density of the Fourier transformed output responses.

Additionally, as [37] used an extended Kalman filter (EKF) to update the parameter values, a comparison is made between the accuracy of the EKF and IMPU method. In [37], the accuracy of only a single parameter estimate is evaluated (with true parameter values $\bar{\boldsymbol{p}}_{\mathrm{ref}} = \left[\bar{k}_{t1,\mathrm{ref}}\ \bar{k}_{t2,\mathrm{ref}}\right]^\top = \left[100800\ 88855\right]^\top$ N/m), of which the absolute relative error is plotted in Fig. 19. To compare, individual IMPU-based estimates of $\bar{\boldsymbol{p}}_{\mathrm{ref}}$ are indicated in Fig. 19 as well. Here, we observe that, already for a relatively low number of training samples (order 100), the IMPU method is competitive with and can even outperform (for sufficiently high $n_s$) the EKF in terms of accuracy. Most likely, this is caused by the fact that the IMPU method simultaneously processes multiple data points (features), yielding a complete overview of all dynamics present in the system,

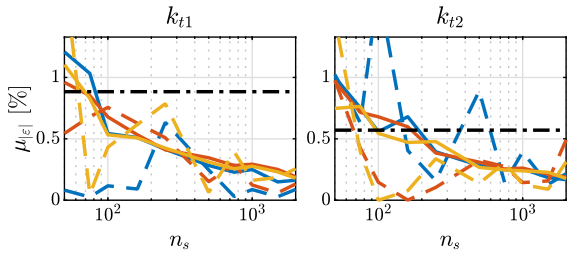


**Fig. 19** Mean absolute relative error over test data set (——) and absolute relative error of estimate of $\boldsymbol{p}_{\mathrm{ref}}$ (- - -) as a function of the number of training samples, $n_s$, per updating parameter of the RPM. The black dash-dotted line (-·-·-) indicates the absolute relative errors of the EKF-based estimates obtained in [37] (independent of $n_s$). The output feature types used for the IMPU method are color-coded: TS (——), MP (——), and RI (——)

and, simultaneously, (partly) filters out the effects of (measurement) noise.

Also, it should be noted that estimating these parameter values takes only 1.5 ms using the IMPU method. Although EKF computation times are not specified in [37], the computation time of the EKF scales significantly with the complexity of the system, in contrast to the inference time of the IMPU method. It is therefore stressed again that the main advantages of the IMPU method over filter-based parameter updating methods (such as EKFs) in the DT context (i.e., online computational efficiency, nonobligatory initial guess, and applicability to simulation models) are of a qualitative nature, see Sect. 1.

## 6 Conclusions and future work

To minimize the mismatch between a physical system and its Digital twin (DT, which is represented as a model), 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 using training samples consisting of output response features, as obtained from a simulated user-defined experiment (with known initial conditions and excitations), and corresponding (known) parameter values. The resulting trained ANN then constitutes a direct inverse mapping between response features and parameter values. In the online phase, this experiment is per-

formed on a real system. Then, the trained ANN is used to map the measured transient-based response features to inferred parameter estimates. These estimates are then used to update the DT. Although data generation and training can take significant time in the offline phase, computation times in the online phase are very small, allowing for (near) real-time parameter updating. 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 (the latter both in the learning and in the inferring phase) of the IMPU approach. Additionally, generalization capabilities are shown and improved by properly choosing output features such that relatively few training samples already achieve satisfactory accuracy or parameter values outside the training space are estimated with reduced but still acceptable accuracy. Furthermore, the parameter updating using the IMPU method is compared to parameter updating using an extended Kalman filter (EKF), where it is observed that, in quantitative sense, the IMPU method is competitive with or even outperforms the EKF.

As estimates of true parameter values that lie beyond the training range become less accurate, quantifying the uncertainty in the estimated parameter values is ongoing research. Additionally, the choice of ANN structure and training settings is far from trivial. Therefore, systematically optimizing the ANN hyperparameters is ongoing research as well. Furthermore, as the IMPU method requires a highly accurate model structure, future work should focus on simultaneous (online) updating of model structure and parameter values. Finally, instead of discussing simulated use cases as done in the current paper, the authors intend to apply the introduced methodology to a physical system in the future.

**Acknowledgements** We would like to thank Dragan Kostić of ASM-PT for the fruitful and inspiring discussions ensuring the practical relevance of the conducted research.

**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 BK, and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

**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.

**Data availability** As the model and (simulated) experiments used to generate all training/test/validation data have 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.

**Declaration**

**Competing Interests** The authors have no relevant financial or non-financial interests to disclose.



## Appendix A: Normalization of parameter values and features

As discussed in Sect. 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, $\boldsymbol{p}_s^{\mathrm{nor}}[j]$, 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:

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

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:

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

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

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

$$\hat{\boldsymbol{p}}_{\bar{s}}[j] = \boldsymbol{p}_{\min}[j] + \hat{\boldsymbol{p}}_{\bar{s}}^{\text{nor}}[j]\left(\boldsymbol{p}_{\max}[j] - \boldsymbol{p}_{\min}[j]\right). \tag{A4}$$

Equivalently, 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:

$$\bar{\boldsymbol{\psi}}_{\bar{s}}^{\text{nor}}[j] = \frac{\bar{\boldsymbol{\psi}}_{\bar{s}}[j] - \boldsymbol{\psi}_{\min}[j]}{\boldsymbol{\psi}_{\max}[j] - \boldsymbol{\psi}_{\min}[j]}. \tag{A5}$$

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

Note that validation samples (features and parameter values) 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

In this appendix, the EoMs of the rigid double beam model as analyzed in Sect. 4 and the EoMs of the roll plane modeling vehicle as used in Sect. 5.

### B.1 Rigid double beam model

The EoMs of the RDB model discussed in Sect. 4 are given by

$$\boldsymbol{M}(\boldsymbol{y})\ddot{\boldsymbol{y}} + \boldsymbol{c}(\boldsymbol{y}, \dot{\boldsymbol{y}}) + \boldsymbol{g}(\boldsymbol{y}) + \boldsymbol{f}(\boldsymbol{y}, \dot{\boldsymbol{y}}) = \boldsymbol{u}, \tag{B1}$$

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

$$\begin{aligned}
\boldsymbol{M}(\boldsymbol{y}) &= \begin{bmatrix} M_{11} & M_{12}(\boldsymbol{y}) \\ M_{21}(\boldsymbol{y}) & M_{22} \end{bmatrix} \\
\boldsymbol{c}(\boldsymbol{y}, \dot{\boldsymbol{y}}) &= \begin{bmatrix} -m_2 \dot{y}_2^2 (l_{\text{CoM},1}\cos(y_2) - l_{\text{CoM},2}\sin(y_2)) \\ 0 \end{bmatrix} \\
\boldsymbol{g}(\boldsymbol{y}) &= \begin{bmatrix} 0 \\ m_2 g l_{\text{CoM},1}\cos(y_2) - m_2 g l_{\text{CoM},2}\sin(y_2) \end{bmatrix} \\
\boldsymbol{f}(\boldsymbol{y}, \dot{\boldsymbol{y}}) &= \begin{bmatrix} k_1(y_1 - l_{k1}) + d_1\dot{y}_1 \\ k_2(y_2 - \theta_{k2}) + d_2\dot{y}_2 \end{bmatrix},
\end{aligned} \tag{B2}$$

where

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

### B.2 Roll plane model

The EoMs of the RPM analyzed in Sect. 5 are given by

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{f}_K(\boldsymbol{q}) + \boldsymbol{f}_C(\dot{\boldsymbol{q}}) = \boldsymbol{f}_U(\boldsymbol{u}), \tag{B3}$$

where explicit dependency on $t$ and $\boldsymbol{p}$ are omitted for brevity. Furthermore, the mass matrix $\boldsymbol{M}$, the vector containing (linear and nonlinear) spring forces $\boldsymbol{f}_K$, the vector with (nonlinear) damper forces $\boldsymbol{f}_C$, and vector with forces due to the prescribed displacement $\boldsymbol{f}_U$ are given by

$$\begin{aligned}
\boldsymbol{M} &= \begin{bmatrix} \frac{m}{2} & \frac{m}{2} & 0 & 0 \\ -\frac{I}{L} & \frac{I}{L} & 0 & 0 \\ 0 & 0 & m_{t1} & 0 \\ 0 & 0 & 0 & m_{t2} \end{bmatrix}, \\
\boldsymbol{f}_K(\boldsymbol{q}) &= \begin{bmatrix} F_{K_1}(q_1 - q_3) + F_{K_2}(q_2 - q_4) \\ \frac{L}{2}\left(F_{K_2}(q_2 - q_4) - F_{K_1}(q_1 - q_3)\right) \\ F_{K_1}(q_3 - q_1) + k_{t1}q_3 \\ F_{K_2}(q_4 - q_2) + k_{t2}q_4 \end{bmatrix}, \\
\boldsymbol{f}_C(\dot{\boldsymbol{q}}) &= \begin{bmatrix} F_{C_1}(\dot{q}_1 - \dot{q}_3) + F_{C_2}(\dot{q}_2 - \dot{q}_4) \\ \frac{L}{2}\left(F_{C_1}(\dot{q}_3 - \dot{q}_1) - F_{C_2}(\dot{q}_4 - \dot{q}_2)\right) \\ F_{C_1}(\dot{q}_3 - \dot{q}_1) \\ F_{C_2}(\dot{q}_4 - \dot{q}_2) \end{bmatrix}, \\
\boldsymbol{f}_U(\boldsymbol{u}) &= \begin{bmatrix} 0 \\ 0 \\ k_{t1}u_1(t) \\ k_{t2}u_2(t) \end{bmatrix},
\end{aligned} \tag{B4}$$

where, for $i = 1, 2$, nonlinear stiffness and damper terms are given by

$$\begin{aligned}
F_{K_i}(q) &= k_i q + k_{i,3} q^3, \\
F_{C_i}(\dot{q}) &= c_i\,(0.2\tanh(10\dot{q})).
\end{aligned} \tag{B5}$$

Furthermore, the prescribed displacements are shown in Fig. 20.

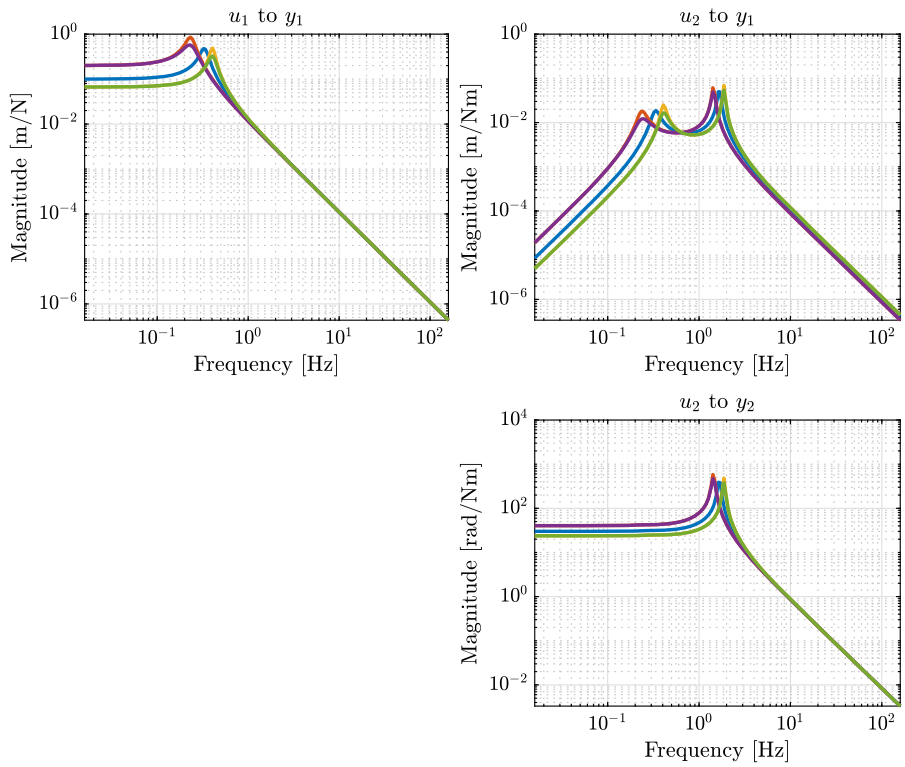


**Fig. 21** Frequency response functions for linearized models for the five parameter cases. The line colors correspond to parameter cases 1 (——), 2 (——), 3 (——), 4 (——), and 5 (——). Please note that the FRF from $u_2$ to $y_1$ is equal to the FRF from $u_1$ to $y_2$

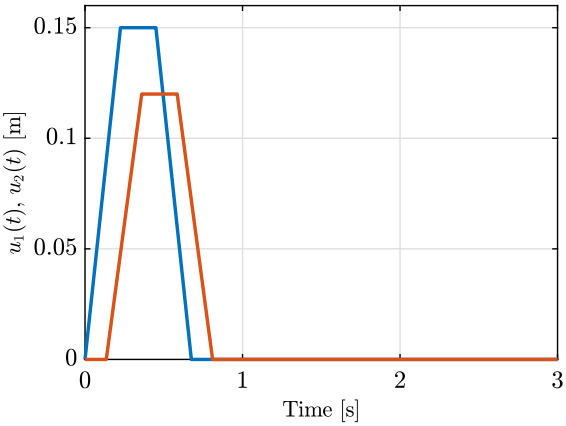


**Fig. 20** Prescribed displacements of the RPM, $u_1(t)$ (——) and $u_2(t)$ (——), representing a speed bump over which is driven with a constant velocity

## Appendix C: Linearized system properties of RDB model

The frequency response functions (FRFs) of the RDB model are plotted in Fig. 21 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:

$$\boldsymbol{\phi}_1 = \begin{bmatrix} 1 \\ -3.80 \times 10^{-2} - 7.88 \times 10^{-3} j \end{bmatrix}$$
$$\boldsymbol{\phi}_2 = \begin{bmatrix} 1.29 \times 10^{-4} + 5.23 \times 10^{-6} j \\ 1 \end{bmatrix}, \tag{C1}$$

with $j$ the imaginary unit.

## References

1. Haag, S., Anderl, R.: Digital twin—proof of concept. Manuf. Lett. **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. Springer, Cham (2017)
3. Glaessgen, E.H., Stargel, D.S.: The digital twin paradigm for future NASA and US air force vehicles. Struct. Dyn. Mater. (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. Eng. Fract. Mech. (2020). https://doi.org/10.1016/j.engfracmech.2019.106766

5. Grieves, M.: Digital twin: manufacturing excellence through virtual factory replication. White Paper **1**, 1–7 (2014)
6. Birk, W., Hostettler, R., Razi, M., Atta, K., Tammia, R.: Automatic generation and updating of process industrial digital twins for estimation and control—a review. Front. Control Eng. **3**(August), 1–20 (2022). https://doi.org/10.3389/fcteg.2022.954858
7. Wright, L., Davidson, S.: How to tell the difference between a model and a digital twin. Adv. Modell. Simul. Engi. Sci. (2020). https://doi.org/10.1186/s40323-020-00147-4
8. Mottershead, J.E., Friswell, M.I.: Model updating in structural dynamics: a survey. J. Sound Vib. **167**(2), 347–375 (1993)
9. Mottershead, J.E., Link, M., Friswell, M.I.: The sensitivity method in finite element model updating: a tutorial. Mech. Syst. Signal Process. **25**(7), 2275–2296 (2011). https://doi.org/10.1016/j.ymssp.2010.10.012
10. Kennedy, M.C., O'Hagan, A.: Bayesian calibration of computer models. J. Royal Stat. Soc. Series B (Stat. Methodol.) **63**(3), 425–464 (2001). https://doi.org/10.1111/1467-9868.00294
11. Viana, F.A.C., Nascimento, R.G., Dourado, A., Yucesan, Y.A.: Estimating model inadequacy in ordinary differential equations with physics-informed neural networks. Comput. Struct. **245**, 106458 (2021). https://doi.org/10.1016/j.compstruc.2020.106458
12. Chiandussi, G., Bugeda, G., Oñate, E.: A simple method for automatic update of finite element meshes. Commun. Numer. Methods Eng. **16**(1), 1–19 (2000). https://doi.org/10.1002/(SICI)1099-0887(200001)16:1<1::AID-CNM310>3.0.CO;2-A
13. 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
14. Hemez, F.M., Doebling, S.W.: Inversion of structural dynamics simulations: state-of-the-art and orientations of the research. Int. Conf. Noise Vib. Eng. **25**, 425–435 (2000)
15. Hemez, F.M., Doebling, S.W.: Review and assessment of model updating for non-linear, transient dynamics. Mech. Syst. Signal Process. **15**(1), 45–74 (2001). https://doi.org/10.1006/mssp.2000.1351
16. Li, W., Chen, Y., Lu, Z.R., Liu, J., Wang, L.: Parameter identification of nonlinear structural systems through frequency response sensitivity analysis. Nonlinear Dyn. **104**(4), 3975–3990 (2021). https://doi.org/10.1007/s11071-021-06481-5
17. Verbeek, G., De Kraker, A., Van Campen, D.H.: Nonlinear parametric identification using periodic equilibrium states—application to an aircraft landing gear damper. Nonlinear Dyn. **7**(4), 499–515 (1995). https://doi.org/10.1007/BF00121110
18. Atalla, M.J., Inman, D.J.: On model updating using neural networks. Mech. Syst. Signal Process. **12**(1), 135–161 (1998). https://doi.org/10.1006/mssp.1997.0138
19. Diaz, M., Charbonnel, P., Chamoin, L.: Robust energy-based model updating framework for random processes in dynamics: application to shaking-table experiments. Comput. Struct. **264**, 106746 (2022). https://doi.org/10.1016/j.compstruc.2022.106746
20. Arora, V., Singh, S.P., Kundra, T.K.: Damped model updating using complex updating parameters. J. Sound Vib. **320**(1–2), 438–451 (2009). https://doi.org/10.1016/j.jsv.2008.08.014
21. Friswell, M.I., Mottershead, J.E., Ahmadian, H.: Finite-element model updating using experimental test data: parametrization and regularization. Philos. Trans. Royal Soc. A: Math. Phys. Eng. Sci. **359**(1778), 169–186 (2001). https://doi.org/10.1098/rsta.2000.0719
22. Kim, K.O., Anderson, W.J., Sandstrom, R.E.: Nonlinear inverse perturbation method in dynamic analysis. AIAA J. **21**(9), 1310–1316 (1983). https://doi.org/10.2514/3.8245
23. Lin, R.M., Zhu, J.: Finite element model updating using vibration test data under base excitation. J. Sound Vib. **303**, 596–613 (2007). https://doi.org/10.1016/j.jsv.2007.01.029
24. 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. J. Sound Vib. **330**(8), 1599–1620 (2011). https://doi.org/10.1016/j.jsv.2010.10.036
25. Modak, S.V., Kundra, T.K., Nakra, B.C.: Comparative study of model updating methods using simulated experimental data. Comput. Struct. **80**(5–6), 437–447 (2002). https://doi.org/10.1016/S0045-7949(02)00017-2
26. Sidhu, J., Ewins, D.J.: Correlation of finite element and modal testing studies of a pracitical structure. In: 2nd International Modal Analysis Conference, Orlando (1984)
27. Berman, A., Nagy, E.J.: Improvement of a large analytical model using test data. AIAA J. **21**(8), 1168–1173 (1983)
28. Caesar, B.: Updating system matrices using modal test data. In: 5th International Modal Analysis Conference, London (1987)
29. 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
30. Ljung, L.: System Identification—Theory for the User, 2nd edn. Pearson, Linköping (1997)
31. Pintelon, R., Schoukens, J.: System Identification: A Frequency Approach, 2nd edn. Wiley, Piscataway (2012)
32. Kerschen, G., Worden, K., Vakakis, A.F., Golinval, J.C.: Nonlinear system identification in structural dynamics: current status and future directions. Soc. Exp. Mech. **21**, 100365 (2007)
33. Schoukens, J., Ljung, L.: Nonlinear system identification: a user-oriented road map. IEEE Control. Syst. **39**(6), 28–99 (2019). https://doi.org/10.1109/MCS.2019.2938121. arXiv:1902.00683
34. 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 (1984). https://doi.org/10.2514/6.1984-929
35. Welch, G., Bishop, G.: An Introduction to the Kalman Filter, Chapel Hill (2001). DOI: https://doi.org/10.1.1.117.6808
36. Lillacci, G., Khammash, M.: Parameter estimation and model selection in computational biology. Comput. Biol. (2010). https://doi.org/10.1371/journal.pcbi.1000696
37. Blanchard, E.: Parameter Estimation Method using an Extended Kalman Filter. In: Proceedings of the Joint North America, Asia-Pacific ISTVS Conference and Annual Meeting of Japanese Society for Terramechanics Fairbanks, Fairbanks (2007)

38. Cheng, M., Becker, T.C.: Performance of unscented Kalman filter for model updating with experimental data. Earthq. Eng. Struct. Dyn. **50**(7), 1948–1966 (2021). https://doi.org/10.1002/eqe.3426
39. Julier, S., Uhlman, J., Durrant-Whyte, H.F.: A new method for the nonlinear transformation of means and covariances in filters and estimators. IEEE Trans. Autom. Control **47**(8), 1406–1408 (2002). https://doi.org/10.1109/TAC.2002.800742
40. Van Der Merwe, R., Wan, E.A.: The square-root unscented Kalman filter for state and parameter-estimation. ICASSP, IEEE Int. Conf. Acoustics Speech Signal Process. Proc. **6**, 3461–3464 (2001). https://doi.org/10.1109/icassp.2001.940586
41. Afshari, H.H., Gadsden, S.A., Habibi, S.: Gaussian filters for parameter and state estimation: a general review of theory and recent trends. Signal Process. **135**, 218–238 (2017). https://doi.org/10.1016/j.sigpro.2017.01.001
42. Kokkala, J., Solin, A., Arkkä, A.S.: Sigma-Point filtering and smoothing based parameter estimation in nonlinear dynamic systems. J. Adv. Inf. Fusion **11**(1), 15–30 (2016). arXiv:1504.06173
43. Arasaratnam, I., Haykin, S.: Cubature Kalman filters. IEEE Trans. Autom. Control **54**(6), 1254–1269 (2009). https://doi.org/10.1109/TAC.2009.2019800
44. Li, S., Yang, Y.: Data-driven identification of nonlinear normal modes via physics-integrated deep learning. Nonlinear Dyn. **106**(4), 3231–3246 (2021). https://doi.org/10.1007/s11071-021-06931-0
45. Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Comput. Methods Appl. Mech. Eng. (2020). https://doi.org/10.1016/j.cma.2019.112789
46. 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. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
47. Zhang, E., Yin, M., Karniadakis, G.E.: Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging. arXiv (2020) arXiv:2009.04525
48. Yan, C.A., Vescovini, R., Dozio, L.: A framework based on physics-informed neural networks and extreme learning for the analysis of composite structures. Comput. Struct. **265**, 106761 (2022). https://doi.org/10.1016/j.compstruc.2022.106761
49. Brunton, S.L., Proctor, J.L., Kutz, J.N., Bialek, W.: Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc. Natl. Acad. Sci. U.S.A. **113**(15), 3932–3937 (2016). https://doi.org/10.1073/pnas.1517384113. arXiv:1509.03580
50. 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) . https://doi.org/10.1098/rspa.2018.0335 arXiv:1711.05501
51. Quade, M., Abel, M., Kutz, J.N., Brunton, S.L.: Sparse identification of nonlinear dynamics for rapid model recovery. Chaos (2018). https://doi.org/10.1063/1.5027470
52. Champion, K., Lusch, B., Kutz, J.N., Brunton, S.L.: Data-driven discovery of coordinates and governing equations. Proc. Natl. Acad. Sci. U.S.A. **116**(45), 22445–22451 (2019). https://doi.org/10.1073/pnas.1906995116. arXiv:1904.02107
53. Kapteyn, M.G., Knezevic, D.J., Willcox, K.: Toward predictive digital twins via component-based reduced-order models and interpretable machine learning. AIAA Scitech Forum Exhibition (2020). https://doi.org/10.2514/6.2020-0418
54. 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. Int. J. Numer. Methods Eng. (2020). https://doi.org/10.1002/nme.6423
55. Kapteyn, M.G., Willcox, K.E.: From physics-based models to predictive digital twins via interpretable machine learning. arXiv (2020) arXiv:2004.11356
56. Lecerf, M., Allaire, D., Willcox, K.: Methodology for dynamic data-driven online flight capability estimation. AIAA J. **53**(10), 3073–3087 (2015). https://doi.org/10.2514/1.J053893
57. Singh, V., Willcox, K.E.: Methodology for path planning with dynamic data-driven flight capability estimation. 17th AIAA/ISSMO Multidiscip. Anal. Optim. Conf. (2016). https://doi.org/10.2514/6.2016-4124
58. Vinnakota, K.C., Bugenhagen, S.M.: Optimization and parameter estimation, genetic algorithms. Encyclopedia Syst. Biol. (2013). https://doi.org/10.1007/978-1-4419-9863-7_291
59. Zimmerman, D.C., Hasselman, T., Anderson, M.: Approximation and identification of nonlinear structural dynamics. Nonlinear Dyn. **39**, 113–128 (2005)
60. Levin, R.I., Lieven, N.A.J.: Dynamic finite element model updating using neural networks. J. Sound Vib. **210**(5), 593–607 (1998)
61. Yong, L., Zhenguo, T.: A two-level neural network approach for dynamic FE model updating including damping. J. Sound Vib. **275**(3–5), 931–952 (2004). https://doi.org/10.1016/S0022-460X(03)00796-X
62. Miller, B.: Application of neural networks for structure updating. Comput. Assist. Mech. Eng. Sci. **18**(3), 191–203 (2011)
63. Neri, R., Arras, M., Coppotelli, G.: FRF-based model updating using neural networks. ISMA **95**, 243–3258 (2016)
64. 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 (2022). https://doi.org/10.1007/978-3-031-04122-8_1
65. 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 (2022). https://doi.org/10.1007/978-3-031-04122-8_1
66. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Cambridge (2006)
67. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
68. Reed, R., Marks, R.J.: Neural Smithing. The MIT Press, Cambridge (1999)
69. Zheng, A.: Evaluating Machine Learning Methods. O'Reilly Media Inc, Sebastopol (2015)

70. 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
71. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. 32nd Int. Conf. Mach. Learn. ICML **2015**(1), 448–456 (2015)
72. Prechelt, L.: Early stopping–But when. Neural Netw. Tricks Trade **32**, 55–70 (1998)
73. MathWorks: Matlab findpeaks function documentation. https://nl.mathworks.com/help/signal/ref/findpeaks.html Accessed 2022-07-09
74. Keesman, K.J., Stigter, J.D.: Optimal parametric sensitivity control for the estimation of kinetic parameters in bioreactors. Math. Biosci. **179**(1), 95–111 (2002). https://doi.org/10.1016/S0025-5564(02)00097-4
75. 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
76. Rakin, A.S., He, Z., Fan, D.: Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recogn. (2019). https://doi.org/10.1109/CVPR.2019.00068
77. 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
78. Keras: Adam. https://keras.io/api/optimizers/adam/ Accessed 2022-06-23
79. Navidi, W.: Statistics for Engineers and Scientists, 3rd edn. McGraw-hill, New York (2006)
80. Huang, P., Yang, X.: Unsupervised feature selection via adaptive graph and dependency score. Pattern Recogn. **127**, 108622 (2022). https://doi.org/10.1016/j.patcog.2022.108622
81. Blanchard, E.D., Sandu, A., Sandu, C., Ahmadian, M., Borggaard, J.T., Leo, D.J.: Polynomial Chaos Approaches to Parameter Estimation and Control Design for Mechanical Systems with Uncertain Parameters. Phd thesis, Virginia Polytechnic Institute and State University (2010)
82. Digital Twin (2022). https://www.digital-twin-research.nl/ Accessed 2022-06-23