Matrix-Free Transfer Function Prediction Using Model Reduction and Machine Learning

Lihong Feng, Peter Benner, Daniele Romano, and Giulio Antonini, *Senior Member, IEEE*

***Abstract*— We propose a novel technique for fast predicting the transfer function of linear systems without information on system matrices by combining machine learning (ML) with model-order reduction. The transfer function of a linear time-invariant system can be written as $H(s) = CG(s)^{-1}B + D$. In some situations, it is difficult to obtain the individual systems matrices that are implicitly included in $G(s)$. The only information available is the data of $G(s)$ and a number of frequency samples s_i. The proposed method derives a reduced-order model (ROM) of the transfer function in the form of neural networks using limited data of $G(s)$. Discrete structure-preserving reduced transfer functions at training samples $s_i, i = 1, \ldots, \ell$, of the frequency are first generated based on the function values of $G(s)$ at those training samples, i.e., $\hat{H}(s_i) = \hat{C}\hat{G}(s_i)^{-1}\hat{B} + D$. An ROM of the original transfer function as a continuous function of the frequency is then learned using the data of the discrete reduced transfer functions. The original transfer function at any testing frequency can then be quickly predicted using either a compact ML model or a deep learning (DL) model with a few layers. The discrete reduced transfer functions are guaranteed to be accurate by a cheap and sharp error estimator, which guarantees the accuracy of the training data for ML. If only the data of the original transfer function are available, then the proposed ML method and DL method can be directly applied without generating the data of the discrete reduced transfer function. The proposed methods are tested on three partial element equivalent circuit (PEEC) models with many delays. The high efficiency and accuracy of the ROM are demonstrated.**

***Index Terms*— Machine learning (ML), model-order reduction (MOR), partial element equivalent circuit (PEEC) method.**

I. Introduction

MODEL-ORDER reduction (MOR) has been active for decades and has demonstrated its robustness and potential for solving complex problems in many fields. Some recent developments can be found in the survey papers and books [1], [2], [3], [4]. Many MOR methods are based on projection [1], [2], [4] and are intrusive. The state-space equation or the corresponding system matrices must be known in order to get the reduced-order model (ROM). On the other hand, the data-driven approaches [3], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24] are nonintrusive and construct the ROM based only on the data of the transfer function or the system solution (snapshots) without querying the system matrices.

Some nonintrusive MOR approaches [3], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] build ROMs in the frequency domain, such as the ROM of the transfer function, while others focus on constructing ROMs of fluid dynamical systems in the time domain [5], [17], [18], [19], [20], [21], [22], [23], [24], [25]. Usually, the time-domain data-driven MOR methods based on snapshots cannot be straightforwardly applied to MOR in the frequency domain if only frequency domain data (or scattering parameters) are available and vice versa, whereas data-driven MOR methods based on the Loewner framework or rational interpolation can build ROMs in both the time domain and the frequency domain, using only the data of the transfer function [3], [7], [14], [15].

In this work, we only consider nonintrusive MOR and propose nonintrusive MOR methods based on the data of the transfer function $H(s) = C(G(s))^{-1}B + D$ or the data of the matrix $G(s)$ associated with the transfer function. Many nonintrusive MOR methods based on the data of the transfer function have been proposed [3], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15]. Most of them construct ROMs of the transfer function, i.e., the reduced transfer function, by assuming the original transfer function being rational, so that it is reasonable to construct the reduced transfer function as a rational function of the frequency. However, there are systems whose transfer functions are nonrational, e.g., time-delay systems, and approximating a nonrational function with a rational function may cause inaccuracy. In [14], the Loewner framework is extended to MOR for systems with a single delay but cannot be extended to systems with multiple delays. The method in [15] considers systems with more than one delay but might be only efficient for small systems with a small number of delays. Rational functions multiplied with delay terms are considered in [6] to approximate the transfer functions of systems with delays, where rational functions are used to approximate nonrational terms.

All the data-driven methods based on the data of the transfer function require some data of the original transfer function via either measurement or black-box simulation. For problems that the data of the original transfer function need to be computed via black-box simulation, large-scale linear systems, such as $G(s)x(s) = B$, must be solved at many frequency samples. This is inefficient for very large systems.

Manuscript received 9 September 2022; revised 16 October 2022; accepted 20 October 2022. Date of publication 29 November 2022; date of current version 12 December 2022. *(Corresponding author: Giulio Antonini.)*

Lihong Feng and Peter Benner are with the Max Planck Institute for Dynamics of Complex Technical Systems, 39106 Magdeburg, Germany (e-mail: feng@mpi-magdeburg.mpg.de; benner@mpi-magdeburg.mpg.de).

Daniele Romano and Giulio Antonini are with the UAq EMC Laboratory, Department of Industrial and Information Engineering and Economics, University of L'Aquila, 67100 L'Aquila, Italy (e-mail: daniele.romano@univaq.it; giulio.antonini@univaq.it).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TMTT.2022.3218854.

Digital Object Identifier 10.1109/TMTT.2022.3218854

Our proposed method includes the following new contributions. First, we propose a machine learning (ML) method based on radial basis functions (RBFs) and a deep learning (DL) method using feed-forward neural networks to learn an ROM of the original transfer function. The original transfer function is allowed to be rational or nonrational. Like other data-driven methods [3], [7], [8], [9], [10], [11], [12], [13], [14], [15], the only required information is the (approximate) data of the original transfer function at limited frequency samples. The proposed methods are easy to implement, and the constructed ML/DL model can quickly predict the original transfer function at testing frequency samples. Second, for large-scale problems that need to compute the data of the original transfer function via black-box simulation, we propose a method for constructing discrete reduced transfer functions based on the data of the matrix $G(s)$ at much fewer training samples than those used for directly constructing the ROM based on the data of the original transfer function. Given the matrix $G(s)$ at any new frequency sample, the discrete reduced transfer at those samples can be quickly computed without solving large linear systems. The data of the discrete reduced transfer functions are then used as training data for constructing the neural network. The accuracy of the discrete reduced transfer functions is guaranteed by an effective error estimator such that the, finally, constructed ROMs are as accurate as those constructed directly using the data of the original transfer function. The proposed methodology is intended for frequency domain analysis. Hence, passivity and causality issues are not addressed in this work.

In Section II, we introduce the problem and propose a method for constructing the discrete reduced transfer functions using error estimation. Section III presents the ML-MOR method based on RBFs and the DL-MOR method. Numerical results for three large partial element equivalent circuit (PEEC) models with potentially hundreds of delays are shown in Section IV and demonstrate the efficiency and accuracy of the proposed methods. Conclusions are given in Section V.

II. Problem Setting and Discrete Reduced Transfer Function

We consider constructing ROMs of transfer functions in the form $H(s) = CG(s)^{-1}B + D$, where $G(s)$ is large and computing the transfer function at many frequency samples is expensive. $G(s)$ may correspond to the standard state-space equation, numerical discretization of time-harmonic Maxwell's equations, time-delay systems, and so on. For standard state-space equations

$$\begin{aligned} Ex(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t). \end{aligned} \tag{1}$$

$G(s) = sE - A$. Here, $x(t) \in \mathbb{R}^n, s = 2\pi \jmath f, f$ is the frequency in Hertz, and $\jmath$ is the imaginary unit. The system matrices are $E, A \in \mathbb{R}^{n\times n}$, while $B \in \mathbb{R}^{n\times n_I}$, $C \in \mathbb{R}^{n_O\times n}$, and $D^{n_O\times n_I}$ are the input, output, and feedthrough matrices, respectively. Here and below, n is called the order of the delay system. For numerically discretized time-harmonic Maxwell's equations with surface losses, e.g.,

$$\begin{aligned} \left(s^2 I + \frac{1}{\sqrt{s}}\tilde{D} + A\right)x(s) &= Bu(t) \\ y(t) &= sB(s). \end{aligned} \tag{2}$$

$G(s) = s^2 I + (1/\sqrt{s})\tilde{D} + A$, where $I \in \mathbb{R}^{n\times n}$ is the identity matrix [26]. For time-delay systems

$$\begin{aligned} \sum_{j=0}^{d} E_j \dot{x}\left(t-\tau_j\right) &= \sum_{j=0}^{d} A_j x\left(t-\tau_j\right) + Bu(t) \\ y(t) &= Cx(t) \quad \forall\ t \geq 0 \end{aligned} \tag{3}$$

we have

$$G(s) = s\sum_{j=0}^{d} E_j e^{-s\tau_j} - \sum_{j=0}^{d} A_j e^{-s\tau_j}.$$

Here, $E_0, \ldots, E_d, A_0, \ldots, A_d \in \mathbb{C}^{n\times n}$ are matrices associated with the delays $0 = \tau_0 < \tau_1 < \cdots < \tau_d$.

The ROMs of the transfer function $H(s)$ can be constructed from the data of $H(s)$ at samples of s. Different from the existing work [3], [7], [8], [9], [10], [11], [12], [13], [14], [15] that construct the ROMs using rational approximation or (extended) Loewner framework, we propose two ML methods for constructing such ROMs in Section II-A. If only the data of $G(s)$ are available, then using the data of the transfer function $H(s)$ to construct the ROMs needs first computing $H(s)$ from $G(s)$ by solving large-scale linear systems $G(s)x(s) = B$ at as many frequency samples as required. In Section II-A, we propose a method for constructing discrete reduced transfer functions at those frequency samples so that the ROMs can be built based on the discrete reduced transfer functions without solving the large-scale linear systems $G(s)x(s) = B$ many times.

A. Constructing Discrete Reduced Transfer Functions

Given only the data of $G(s)$ and the matrices B, C, and D, we propose a method of constructing discrete reduced transfer functions at sampled frequencies. The idea is motivated by the projection-based intrusive MOR methods [27]. In fact, the greedy algorithm for nonparametric systems proposed in [27] can be used to construct a projection matrix $V \in \mathbb{R}^{n\times r}, r \ll n$. Given any frequency sample s_0, a reduced transfer function data $\hat{H}(s_0)$ can be obtained as $\hat{H}(s_0) = CV(V^T G(s_0)V)^{-1}V^T B + D$, where only a small linear system $V^T G(s_0)V\hat{x}(s_0) = V^T B$ needs to be solved. If $\hat{H}(s_0)$ is accurate enough, then $\hat{H}(s_0)$ can safely replace the original data $H(s_0)$ and can be used as the training data for constructing the ROM of $H(s)$. Fortunately, this is guaranteed by the effective error estimator Δ_1^{pr} proposed in [27].

We modify the algorithm for nonparametric systems introduced in [27] by assuming that, except for the input, output, and feedthrough matrices B, C, and D, the only available data are $G(s)$ at a limited number of samples of s, i.e., $G(s_i), i = 1, \ldots, m, m \ll n$. Algorithm 1 is the modified algorithm.

Note that, during iterations of Algorithm 1, the large-scale system $G(s_k)x(s_k) = B$ (with possibly multiple right-hand

Algorithm 1 Computing a Frequency-Independent Projection Matrix $V \in \mathbb{R}^{n\times r}$ Based on the Data of $G(s)$

Input: $G(s_i), i = 1, \ldots, m, m \ll n$ and the input, output, feedthrough matrices B, C, D. Ξ: a set consisting of the m samples $s_i, i = 1, \ldots, m$. Error tolerance $\varepsilon < 1$.
Output: Projection matrix $V \in \mathbb{R}^{n\times r}$
1: Choose initial frequency $s_1 \in \Xi$. $\texttt{err} = 1$, $k = 1$.
2: $V \leftarrow \emptyset$.
3: **while** $\texttt{err} > \varepsilon$ **do**
4: $V_{s_k} = G^{-1}(s_k)B$, expand V by $V = \text{orth}\{V, V_{s_k}\}$.
5: Compute s_{k+1} such that $s_{k+1} = \arg\max_{s\in\Xi} \Delta(s)$.
6: $\texttt{err} = \Delta(s_{k+1})$.
7: **end while**

side vectors for multiple input systems) must also be solved at s_k in order to compute V_{s_k}. However, we emphasize that it is only solved once at each iteration step at the single selected sample $s_k = \arg\max_{s\in\Xi} \Delta(s)$. Usually, the algorithm converges in a few iterations, whereas using only the data of $H(s)$ at a few samples of s is not enough to construct an accurate ROM of $H(s)$, and the data-driven methods based on the data of $H(s)$ would need many more solutions of a large system.

Remark 1: It is known that projection-based MOR using a projection matrix V is very efficient for problems with low-dimensional solution spaces. Here, we focus on problems with low-dimensional solution spaces. Algorithm 1 is used to compute a projection matrix V whose columns are some basis vectors of the solution space. At each iteration, Algorithm 1 computes one new basis vector of the solution space. If the solution space is of low dimension, it can be accurately represented by only a few basis vectors. Then, Algorithm 1, finally, converges after computing those few basis vectors. The stopping criteria for Algorithm 1 are based on the proposed error estimation, which efficiently measures the error between the original solution and the reduced solution represented by the basis vectors.

The next concern is computing the error estimator $\Delta(s)$ at all the samples $s_i, i = 1, \ldots, m$. To show this, we present some details of computing $\Delta(s)$. Here, we employ the error estimator Δ_1^{pr} from [27] since it is shown to be most effective among the error estimators analyzed in [27]. It estimates the error of $\hat{H}(s)$ in the matrix max-norm, i.e.,

$$\|H(s) - \hat{H}(s)\|_{\max} := \max_{i,j} |H_{i,j}(s) - \hat{H}_{i,j}(s)|$$

and is defined as follows:

$$\|H(s) - \hat{H}(s)\|_{\max} \approx \|C\tilde{X}_r(s)\|_{\max} =: \Delta(s).$$

Without causing ambiguity, we have removed the superscript "pr" and the subscript 1 from $\Delta_1^{pr}(s)$. Here, $\tilde{X}_r(s) = V_r Z_r(s)$, and $Z_r(s)$ is the solution to a reduced primal-residual system defined as

$$V_r^T G(s) V_r Z_r(s) = V_r^T R_{pr}(s) \tag{4}$$

where $R_{pr}(s) = B - G(s)\hat{X}_{pr}(s)$. The quantity $\hat{X}_{pr}(s) := VZ_{pr}(s)$, and $Z_{pr}(s)$ is the solution to the reduced primal system defined as

$$\hat{G}(s) Z_{pr}(s) = V^T B \tag{5}$$

where $\hat{G}(s) = V^T G(s) V$. From (4), it is seen that computing $\Delta(s)$ at any sample $s_k \in \Xi$ needs to compute another projection matrix V_r and get $\tilde{X}_r(s) = V_r Z_r(s)$. V_r can be computed similarly as V but using samples different from those used for computing V. Detailed analysis can be found in [27]. Finally, Algorithm 1 is modified to Algorithm 2 that includes computing V_r for $\Delta(s)$.

Algorithm 2 Computing a Frequency-Independent Projection Matrix $V \in \mathbb{R}^{n\times r}$ Based on the Data of $G(s)$ and Using an Estimator in [27]

Input: $G(s_i), i = 1, \ldots, m, m \ll n$, and the input, output and feedthrough matrices B, C, D. Ξ: a set consisting of the m samples $s_i, i = 1, \ldots, m$. Error tolerance $\varepsilon < 1$.
Output: Projection matrix $V \in \mathbb{R}^{n\times r}$
1: Choose an initial $s_1 \in \Xi$, choose another frequency sample $s_1^r \in \Xi$ and $s_1^r \neq s_1$. $\texttt{err} = 1$, $k = 1$.
2: $V \leftarrow \emptyset$, $V_r \leftarrow \emptyset$.
3: **while** $\texttt{err} > \varepsilon$ **do**
4: $V_{s_k} = G^{-1}(s_k)B$, expand V by $V = \text{orth}\{V, V_{s_k}\}$.
5: $V_{s_k}^r = G^{-1}(s_k^r)B$, expand V by $V_r = \text{orth}\{V, V_r, V_{s_k}^r\}$.
6: Compute s_{k+1} such that $s_{k+1} = \arg\max_{s\in\Xi} \Delta(s)$.
7: Compute s_{k+1}^r such that $s_{k+1}^r = \arg\max_{s\in\Xi} \epsilon_r(s)$. $\epsilon_r(s) = \max_{j\le n_I} \|R_j^r(s)\|_2$, R_j^r is the j-th column of $R^r(s)$, and $R^r(s) = R_{pr}(s) - G(s)\tilde{X}_r(s)$.
8: $\texttt{err} = \Delta(s_{k+1})$.
9: **end while**

We summarize the computation of the discrete reduced transfer functions using Algorithm 2 as follows.

1) Solve $G(s)x(s) = B$ at two samples of s at each iteration.
2) Compute the error estimator $\Delta(s)$ at all $s \in \Xi$ at each iteration.
3) Upon convergence, obtain an s-independent matrix V.
4) At any new sample s^* outside of Ξ and given $G(s^*)$, compute the reduced transfer function $\hat{H}(s^*) = \hat{C}\hat{G}(s^*)^{-1}\hat{B} + D$ without solving a large-scale linear system at s^*. Here, $\hat{C} = CV \in \mathbb{R}^{n_O\times r}$, $\hat{G}(s^*) = V^T G(s^*) V \in \mathbb{R}^{r\times r}$, $\hat{B} = V^T B \in \mathbb{R}^{r\times n_I}$.

Furthermore, Algorithm 2 shows that the posterior error estimators proposed in [27] are applicable to systems lacking information on system matrices.

In order to compute $\Delta(s)$ in Algorithm 2, the residuals $R_{pr}(s)$ and $R_r(s)$ must be computed for all m samples in the training set Ξ. The computational complexity is $O(n^2)$, which depends on n. However, for some applications, this is still less than the computational complexity of solving an $n \times n$ large-scale linear system, e.g., the time-delay system considered in Section IV in this work, where $G(s)$ is not sparse.

Although an additional large system $G(s_k^r)x(s_k^r) = B$ needs to be solved at each iteration of Algorithm 2, the overall number of large system computations is still less than directly using the data of $H(s)$ to construct the ROM.

This can be seen from the examples in the numerical tests in Section IV.

It is, however, not practical to use $\hat{H}(s)$ as an ROM of the original transfer function since $\hat{H}(s)$ needs $G(s) \in \mathbb{C}^{n\times n}$ to be available at any value of the frequency. In fact, it is only possible to derive $G(s)$ at a limited number of frequency samples, as storing all large-scale matrices $G(s_k)$ becomes more and more difficult when the dimension n is very large. This is also the motivation for using Algorithm 4 (to be introduced in Section III) to generate the data of the transfer function, where $G(s)$ is computed on demand, rather than being precomputed at many frequency samples. However, if considering $\hat{H}(s)$ as the ROM and allowing that $G(s)$ is generated online in order to compute $\hat{H}(s)$, it is still time-consuming, especially for some frequency analyses where $G(s)$ needs to be generated many times at many frequency samples. Therefore, we propose to use neural networks to construct an ROM of the original transfer function so that the ROM allows fast simulation and is compact to be easily stored. The discrete reduced transfer functions $\hat{H}(s_k)$ at a limited number of frequency samples $s_k, k = 1, \ldots, \ell$, are only used to train the neural network to avoid solving many large-scale linear systems.

III. ROM Construction Using ML/DL

Methods of constructing an ROM of the original transfer function using limited data of the transfer function are proposed in this section. The first method is based on RBF interpolation. The second method is a DL method using fully connected deep feedforward neural networks (DFNNs).

A. Data Generation

In this work, we only consider data obtained from modeling and simulation rather than from measurements though the proposed ML MOR methods in this section could also be straightforwardly applied to measured data. The data of $G(s)$ are obtained by modeling and simulating PEEC models with delays. When the explicit form in (3) is available, we use Algorithm 3 to generate the data of the original transfer function and the data of the discrete reduced transfer function.

In general, a delayed PEEC model is difficult to be accurately described in the form of (3) with system matrices E_i and A_i being explicitly available. The system matrices E_i and A_i are only quantities used to approximate $G(s)$, and often, they are difficult to obtain. Instead, modeling $G(s)$ as a black box is much easier. In this situation, we describe in Algorithm 4 the process of generating the data of $H(s)$ or $\hat{H}(s)$ assuming that only a black-box model of $G(s)$ is available. Considering that $G(s)$ is very large, it is not possible to precompute $G(s)$ at many frequency samples, which will occupy too much storage space. The only practical way is to compute $G(s)$ on demand and store only the data of the transfer function. Algorithm 4 exactly follows this way to compute the data of the transfer functions.

From Algorithms 3 and 4, we see that, when the number m of frequency samples in the training set Ξ of Algorithm 2 is much smaller than the number ℓ of the data samples, then computing the discrete reduced transfer functions $\hat{H}(s_i), i = 1, \ldots, \ell$, should be much faster than computing the data of the original transfer function $H(s_i), i = 1, \ldots, \ell$. This is often the case for problems whose solution spaces are of low dimensions, for example, the examples in the numerical tests.

Algorithm 3 Generating Data of $H(s)$ or $\hat{H}(s)$ Based on (3)

Input: System matrices in (3).

Output: Data of $H(s)$ or $\hat{H}(s)$ at frequency samples $s_1, \ldots, s_\ell$

1: **if** Data of $H(s)$ are required **then**
2: For each frequency sample $s_i, i = 1, \ldots, \ell$, assemble $G(s_i)$ with the system matrices $E_j, A_j, j = 1, \ldots, d$ (3), then solve a linear system of dimension n with n_I right-hand-sides: $G(s_i)X(s_i) = B$ and compute $H(s_i)$: $H(s_i) = CX(s_i) + D$.
3: **else if** Data of $\hat{H}(s)$ are required **then**
4: For each frequency sample s_k in the training set Ξ of Algorithm 2, assemble $G(s_k)$ from the system matrices $E_j, A_j, j = 1, \ldots, d$ (3).
5: Implement Algorithm 2 and get $V \in \mathbb{R}^{n\times r}$, precompute $\hat{E}_j = V^T E_j V$, $\hat{A}_j = V^T A_j V, j = 1, \ldots, d$, $\hat{C} = CV$, $\hat{B} = V^T B$.
6: For each frequency sample $s_i, i = 1, \ldots, \ell$, assemble $\hat{G}(s_i)$ with the small system matrices $\hat{E}_j, \hat{A}_j$ following the expression of $G(s)$ in (3), then solve a linear system of dimension $r \ll n$ with n_I right-hand-sides: $\hat{G}(s_i)\hat{X}(s_i) = \hat{B}$.
7: Compute $\hat{H}(s_i)$: $\hat{H}(s_i) = \hat{C}\hat{X}(s_i) + D$.
8: **end if**

Algorithm 4 Generating Data of $H(s)$ or $\hat{H}(s)$ Based on the Data of $G(s)$

Input: Model of $G(s)$, input, output and feedthrough matrices B, C, D

Output: Data of $H(s)$ or $\hat{H}(s)$ at frequency samples $s_1, \ldots, s_\ell$

1: **if** Data of $H(s)$ are required **then**
2: For each frequency sample $s_i, i = 1, \ldots, \ell$, simulate the model of $G(s)$ at s_i, and get $G(s_i)$, then solve a linear system with n_I right-hand-sides: $G(s_i)X(s_i) = B$, and compute $H(s_i)$: $H(s_i) = CX(s_i) + D$.
3: **else if** Data of $\hat{H}(s)$ are required **then**
4: For each frequency sample s_k in the training set Ξ of Algorithm 2, simulate the model of $G(s_k)$.
5: Implement Algorithm 2, and get V and precompute $\hat{C} = CV$, $\hat{B} = V^T B$.
6: For each frequency sample $s_i, i = 1, \ldots, \ell$, simulate the model of $G(s_i)$, and compute $\hat{G}(s_i) = V^T G(s_i) V$, then solve a small linear system with n_I righ-hand-sides: $\hat{G}(s_i)\hat{X}(s_i) = \hat{B}$.
7: Compute $\hat{H}(s_i)$: $\hat{H}(s_i) = \hat{C}\hat{X}(s_i) + D$.
8: **end if**

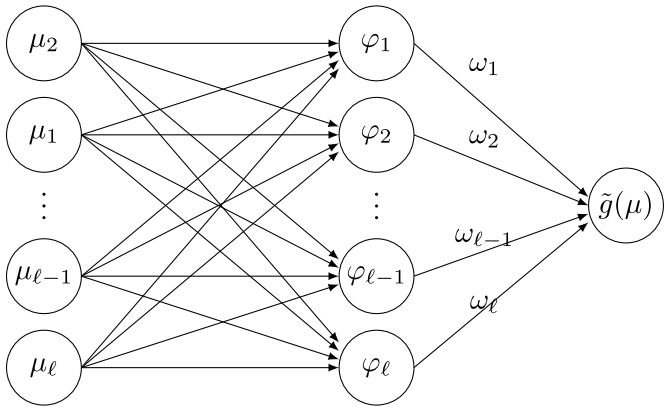


Fig. 1. RBF interpolation as a shallow neural network.

B. ROM in the Form of Shallow Neural Networks

Interpolating a scalar-valued function $g(\mu)$ using RBFs is a well-known ML method, and the model learned is a neural network with a single layer. $g(\mu)$ is approximated as

$$g(\mu) \approx \tilde{g}(\mu) := \sum_{j=1}^{\ell} \omega_j \varphi(\mu - \mu_j). \tag{6}$$

Enforcing interpolation between the RBF approximation $\tilde{g}(\mu)$ and $g(\mu)$ at μ_j, we obtain

$$\begin{bmatrix} \varphi(\mu_1-\mu_1) & \dots & \varphi(\mu_1-\mu_\ell) \\ \vdots & \vdots & \vdots \\ \varphi(\mu_\ell-\mu_1) & \dots & \varphi(\mu_\ell-\mu_\ell) \end{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_\ell \end{bmatrix} = \begin{bmatrix} g(\mu_1) \\ \vdots \\ g(\mu_\ell) \end{bmatrix}. \tag{7}$$

Here, $\mu_j, j = 1, \dots, \ell$, are the training samples and $g(\mu_j)$, $j = 1, \dots, \ell$, are the training data used to train the ML model on the right-hand side of (6). The weights $\omega_j, j = 1, \dots, \ell$, can be easily obtained from (7) or a regularized version thereof (in case the coefficient matrix is ill-conditioned) [28]. Fig. 1 illustrates the RBF interpolation as a shallow neural network, where $\varphi_i := \varphi(\mu - \mu_i)$, $i = 1, \dots, \ell$. The centers μ_i constitute the input layer, and the single hidden layer is composed of the weights and the activation function $\varphi(\mu - \mu_i)$ with shifts $\mu_i, i = 1, \dots, \ell$. The output layer is $\tilde{g}(\mu)$, an approximation of $g(\mu)$.

A list of commonly used RBFs can be found in [29]. In this work, we use the inverse multiquadratic function

$$\varphi(\mu - \mu_j) = \frac{1}{1 + (\sigma \|\mu - \mu_j\|_2)^2}$$

where the constant variable σ is the so-called shape parameter. Since all the RBFs are real-valued, and when $\tilde{g}(\mu)$ is used to approximate the transfer function with complex values, we take $\mu := f$, the ordinary frequency, and $g(\mu) := H_{ij}^R(2\pi \jmath f)$ or $g(\mu) := H_{ij}^I(2\pi \jmath f)$. Here, $H_{ij}^R(2\pi \jmath f)$ and $H_{ij}^I(2\pi \jmath f)$ are the real and imaginary parts of the ijth entry of $H(s)$, respectively, $i = 1, \dots, n_O$, $j = 1, \dots, n_I$. Note that $s = 2\pi \jmath f$. Each entry of $H(s)$ is approximated by RBF interpolation, and the real part and the imaginary part are approximated separately. For systems with n_I inputs and n_O outputs, we construct $2n_I n_O$ neural networks via RBF interpolation in the form of $\tilde{g}(f)$. Each neural network has a single layer. Once the neural networks are trained, i.e., once the weights are computed via (7), they are ready for use to predict the original transfer function at any new frequency sample. We summarize the above steps as Algorithm 5.

Algorithm 5 RBF-ROM Construction From the Data of $H(s)$

Input: $H(2\pi \jmath f_k), k = 1, \dots, \ell, \ell \ll n$

Output: the ROM $\tilde{H}(f) \in \mathbb{R}^{n_O \times n_I}$ in the form of neural networks

1: $i = 1, j = 1$.
2: **while** $i, j \le n_O, n_I$ **do**
3: Let $g(\mu_k) = H_{ij}^R(2\pi \jmath f_k)$, $k = 1, \dots \ell$, and solve (7) to get the corresponding weights ω_k^R, form the neural network $\tilde{H}_{ij}^R(f) = \sum_{k=1}^{\ell} \omega_k^R \Phi(f - f_k)$.
4: Let $g(\mu_k) = H_{ij}^I(2\pi \jmath f_k)$, $k = 1, \dots \ell$, and solve (7) to get the corresponding weights ω_k^I, form the neural network $\tilde{H}_{ij}^I(f) = \sum_{k=1}^{\ell} \omega_k^I \Phi(f - f_k)$.
5: Let $\tilde{H}_{ij}(f) = \tilde{H}_{ij}^R(f) + \jmath \tilde{H}_{ij}^I(f)$ be the ij-th entry of $\tilde{H}(f)$.
6: **end while**

Algorithm 5 constructs an ROM $\tilde{H}(s)$ that is composed of $2n_O \times n_I$ shallow neural networks built by RBF interpolation. Since, usually, $\ell \ll n$, solving an $\ell \times \ell$ linear system in (7) is very fast. When $n_O, n_I \ll n$, Steps 3 and 4 can be finished for all i, j pairs in a very short time. This means that the neural networks can be quickly trained. Algorithm 5 considers deriving an ROM $\tilde{H}_{ij}(s)$ using the data of $H(s)$. When only the data of $G(s)$ are available, then using Algorithm 5 will inevitably involve solving l large-scale linear systems $G(s_k)x(s_k) = B, k = 1, \dots, \ell$, in order to get the data of $H(s_k)$; see Step 3 of Algorithm 4. This will slow down the training process. Furthermore, once the ℓ samples are found to be insufficient or improper, more large linear systems need to be solved. To reduce the computational cost in solving large linear systems, we propose to replace the data of the original transfer function with the discrete reduced transfer functions computed with Algorithm 2.

In the following, we present Algorithm 6 that computes an ROM of the original transfer function based on the data of discrete reduced transfer functions at ℓ frequency samples. Algorithm 4 is used to generate the data (Steps 4–7), where Algorithm 2 is employed to compute the projection matrix V. The data are then used in Algorithm 6 to compute the weights in (7). If $\hat{H}(s)$ at more frequency samples needs to be computed, then a small $r \times r$ linear system $V^T G(s^*) V \hat{X}(s^*) = B$ needs to be solved at any new sample s^*.

We notice from Algorithm 2 that there are still two large linear systems to be solved at each iteration in order to compute V. However, the total number of large linear systems solved in Algorithm 2 is still much less than ℓ, the number of large linear systems to be solved when using Algorithm 5. Finally, given the data of $G(s)$ only, Algorithm 6 should be more efficient than Algorithm 5 if $\hat{H}(2\pi \jmath f_k)$ are accurate substitutes of the original data $H(2\pi \jmath f_k), k = 1, \dots, \ell$.

This can be guaranteed by the sharp error estimator $\Delta(s)$ used in Algorithm 2.

Algorithm 6 RBF-ROM Construction From Data of the Discrete Reduced Transfer Functions

Input: discrete reduced transfer functions $\hat{H}(s_i), i = 1, \ldots, \ell, \ell \ll n$

Output: the ROM $\tilde{H}(f) \in \mathbb{R}^{n_O \times n_I}$ in the form of neural networks

1: $i = 1, j = 1$.
2: **while** $i, j \leq n_O, n_I$ **do**
3: Let $g(\mu_k) = \hat{H}^R_{ij}(2\pi \jmath f_k)$, $k = 1, \ldots \ell$, and solve (7) to get the corresponding weights ω^R_k, form the neural network $\tilde{H}^R_{ij}(f) = \sum_{k=1}^{\ell} \omega^R_k \Phi(f - f_k)$.
4: Let $g(\mu_k) = \hat{H}^I_{ij}(2\pi \jmath f_k)$, $k = 1, \ldots \ell$, and solve (7) to get the corresponding weights ω^I_k, form the neural network $\tilde{H}^I_{ij}(f) = \sum_{k=1}^{\ell} \omega^I_k \Phi(f - f_k)$.
5: Let $\tilde{H}_{ij}(f) = \tilde{H}^R_{ij}(f) + \jmath \tilde{H}^I_{ij}(f)$ be the ij-th entry of $\tilde{H}(f)$.
6: **end while**

C. ROM in the Form of Deep Neural Networks

This section explores the capability of DL in building an ROM of the original transfer function. We propose to use fully connected DFNNs to represent the ROM. In particular, we use DFNN to learn each entry $H_{ij}(s)$ of the original transfer function; the final ROM of $H(s)$ is in the form of $n_O \times n_I$ DFNNs.

The input layer of each DFNN is composed of the frequency samples f_k, and the 2-D outputs are the real and imaginary parts of the ROM, respectively. To the best of our knowledge, the activation functions used in DL are all real-valued; therefore, we need to use DFNN to learn the real part and the imaginary part of $H_{ij}(s)$ separately, as is done by the RBF interpolation. Unlike RBF interpolation where two neural networks are necessary to separately represent the real and the imaginary parts, a single DFNN is sufficient to learn both the real part and the imaginary part.

DFNN for learning each entry $H_{ij}(s), \forall\, 0 < i < n_O, 0 < j < n_I$, of the transfer function is illustrated in Fig. 2. Here, we use $\tilde{H}_{ij}$ to indicate the approximate transfer function learned by the DFNN. The output layer is 2-D with two columns: one is composed of the real part $h^R(f_k)$ of $\tilde{H}_{ij}$ at the input frequency samples, and the other is composed of the imaginary part $h^I(f_k)$ at those samples. For each entry $H_{ij}(s)$, we train a DFNN and use the trained DFNN to predict $H_{ij}(s)$ at any new frequency sample.

If only the data of the original transfer function are available, the loss function is defined as the mean square error (mse) between the data of the transfer function and the outputs of the DFNN, i.e.,

$$\text{mse}_H = \frac{1}{\ell} \sum_{k=1}^{\ell} \Big(|H^R_{ij}(2\pi \jmath f_k) - h^R(f_k)|^2 + \left| H^I_{ij}(2\pi \jmath f_k) - h^I(f_k) \right|^2 \Big).$$

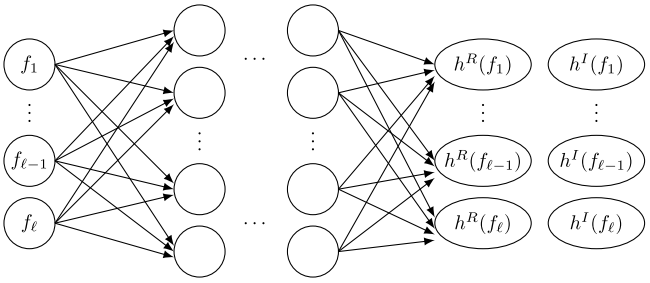


Fig. 2. DFNN for learning the transfer function.

If the data of the discrete reduced transfer function are available, then the mse is defined as the difference between the discrete reduced transfer function $\hat{H}(2\pi \jmath f_k)$ and the outputs of the DFNN, i.e.,

$$\text{mse}_G = \frac{1}{\ell} \sum_{k=1}^{\ell} \Big(\left| \hat{H}^R_{ij}(2\pi \jmath f_k) - h^R(f_k) \right|^2 + \left| \hat{H}^I_{ij}(2\pi \jmath f_k) - h^I(f_k) \right|^2 \Big).$$

For different examples and/or different entries of the transfer function, DFNN may need different hidden layers and activation functions. We list the detailed information on DFNN for each example in Section IV.

IV. Numerical Results

We test the efficiency of the proposed methods for three PEEC models with potentially many delays. We propose to use RBF interpolation or DFNN to generate the ROM of the original transfer function.

For the first model, explicit delays and all the matrices associated with the delays have been extracted. Therefore, both intrusive and nonintrusive MOR methods can be applied. In order to show the robustness of the proposed nonintrusive MOR methods, we use Algorithm 3 to generate the data of the original transfer function at $\ell \ll n$ frequency samples and the data of the discrete reduced transfer function at the same frequency samples. We also use this example and RBF interpolation to compare the results based on discrete reduced transfer functions (see Algorithm 6) with the results based on the data of the original transfer function (see Algorithm 5).

The only available system information for the second example is the sample of $G(s)$ and the input and output matrices. $G(s)$ can be computed at any frequency sample from a black-box model. Given the large scale of the matrix, $G(s)$ at many frequency samples cannot be precomputed and stored. Therefore, we use Algorithm 4 to generate the data of the transfer functions, where $G(s)$ is computed online and only the data of the transfer function are stored for use.

The third example is a model of a 50-cm-long transmission line. Similarly, we only have the data of $G(s)$ for this model, from which we compute the data of the transfer function. Only the data of the transfer function need to be stored for training and testing the proposed ML methods.

After the data of the transfer function are obtained, they are ready to be used for training the RBF network or the DFNN.

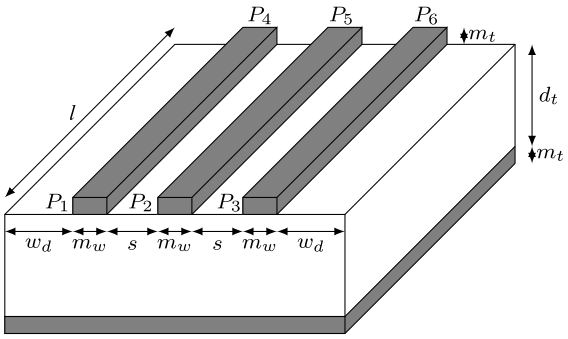


Fig. 3. Three coplanar microstrips.

TABLE I
COPLANAR MODEL. RBF NETWORK PERFORMANCE COMPARISON: USING DATA OF $H(s)$ VERSUS USING DATA OF $\hat{H}(s)$

ℓ	200		334
Algorithm	5	6	6
n_{LS}	200	22	22
n_{SS}	0	200	334
$\text{Error}_{\max} H_{11}^{\text{RBF}}$	$4.8 \cdot 10^{-1}$	$4.8 \cdot 10^{-1}$	$7.8 \cdot 10^{-4}$
$\text{Error}_{\max} H_{43}^{\text{RBF}}$	$1.36 \cdot 10^{-1}$	$1.36 \cdot 10^{-1}$	$4 \cdot 10^{-3}$
$\text{Error}_{\max} H_{62}^{\text{RBF}}$	$1.33 \cdot 10^{-1}$	$1.33 \cdot 10^{-1}$	10^{-3}
Runtime [s]	0.72	0.64	1.04

The error in the figures is defined as

$$\text{Error}(s) = \frac{\left|\tilde{H}_{ij}(s) - H_{ij}(s)\right|}{\max\limits_{s \in \Xi_{\text{test}}} \left|H_{ij}(s)\right|}$$

for a particular entry $H_{ij}(s)$ of $H(s)$. Ξ_{test} is the set of testing frequency samples with cardinality $|\Xi|_{\text{test}} > \ell$. $\text{Error}_{\max}$ in the tables is

$$\text{Error}_{\max} = \max_{s \in \Xi_{\text{test}}} \text{Error}(s).$$

A. Test 1: Coplanar Microstrips PEEC Model

The first example is a PEEC model of three coplanar microstrips sketched in Fig. 3. The length of each strip is $l = 5$ cm, the thickness of the dielectric is $d_t = 0.8$ mm, and the spacing between two strips is $s = 0.3$ mm. The width of each strip is $m_w = 0.178$ mm. The thickness of each strip and that of the ground plane are both $m_t = 0.035$ mm. The left and right wings of the microstrips are both with width $w_d = 3$ mm. The relative dielectric constant is set to be $\varepsilon_r = 4$, and the conductivity of the metal is assumed to be $\sigma = 5.8 \times 10^7$ S/m. The six ports, located between the ends of each strip and the ground plane below, are terminated on load resistors $R_{\text{load}} = 50\ \Omega$. The frequency band of interest is $[0, 10]$ GHz. The original model has degrees of freedom $n = 16\,644$ with $n_I = 6$ inputs and $n_O = 6$ outputs. There are $d = 168$ delays. We are interested in using the ROM of the transfer function to predict the original transfer function at $|\Xi_{\text{test}}| = 1000$ frequency samples. We pick three entries of the original transfer function to show the robustness of the proposed RBF-ROM and DFNN-ROM. Those entries have more resonant peaks and are more difficult to be approximated than others. The simulation results for both these methods are shown in Fig. 4 where Ref denotes the reference solution obtained with the PEEC method. Details are provided in the following discussions.

1) Discussion on the Results of RBF Interpolation: The results of RBF interpolation using the data of the discrete reduced transfer function, i.e., the results of Algorithm 6, are shown in Fig. 4. The cardinality m of the training set in Algorithm 2 is set as $m = 30$, i.e., only data of $G(s)$ at 30 frequency samples are used to construct the projection matrix V. Since Algorithm 2 converges in 11 iterations, $2 \times 11 = 22$ large linear systems (with $n_I = 6$ right-hand side vectors) of dimension $n = 16\,644$ are solved to build the final projection matrix V; see Steps 4 and 5. Then, V is used to compute the discrete reduced transfer functions at $\ell = 200$ training frequency samples, which needs the data of $G(s)$ at the same samples. Algorithm 5 using the data of the original transfer function at the same 200 frequency samples produces results with very similar accuracy. However, many more large linear systems must be solved to get the data.

In Table I, we compare the performance of Algorithm 6 to that of Algorithm 5 in more detail. Algorithm 5 needs to compute the data of the original transfer function by solving $n_{\text{LS}} = 200$ large-scale systems of dimension $n = 16\,644$, whereas Algorithm 6 computes 200 discrete reduced transfer functions by solving only $n_{\text{LS}} = 22$ large-scale systems and $n_{\text{SS}} = 200$ small systems of dimension $r = 132$. For the three entries of $H(s)$, the two algorithms produce the same maximal error $\text{Error}_{\max}$. After the training data are computed, the RBF network can be trained, and the trained RBF-ROM can be used as the ROM of $H_{ij}(s)$ to compute the approximate transfer function H_{ij}^{RBF} at 1000 frequency samples. The runtime in the table includes the RBF network training time and the time of computing the approximate transfer function H_{ij}^{RBF} at 1000 frequency samples. The runtimes for different entries of $H(s)$ are almost the same, and we show only the runtime for a single entry. It is seen that the RBF ML approach is very cheap to be trained and efficient for prediction. The time spent on training and prediction together is less than 1 s. The error is less than $1.4 \cdot 10^{-1}$. From Fig. 4, we see that the RBF-ROM reproduces both the magnitudes and the angles of the original transfer functions pretty well.

It is possible to further improve the accuracy of RBF-ROM by using more training data. If $\ell = 334$ discrete transfer functions or the data of the original transfer function at 334 frequency samples are used to train the RBF network, then the RBF-ROM gives results that overlap with those of the original transfer function and no difference can be seen from the plots. The runtime including training and predicting is still very short. To avoid repetition, we do not show the figures but only list the results of Algorithm 6 in Table I. Once the matrix V is computed, the process of solving small linear systems to get the discrete reduced transfer functions is very fast. Taking the 334 training samples as an example, the total

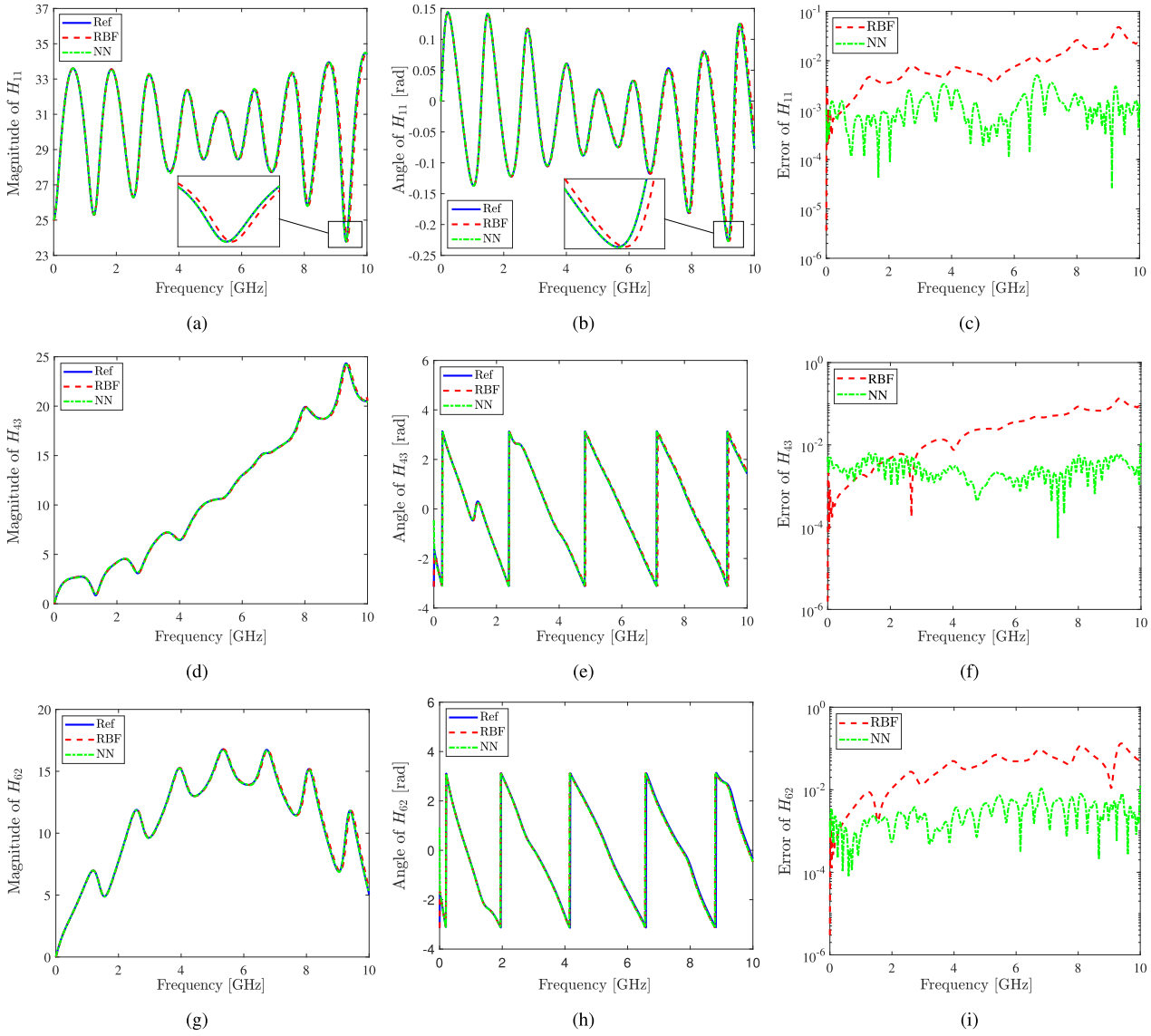


Fig. 4. Experimental results for the coplanar microstrips example. (a) Magnitude of H_{11}. (b) Angle of H_{11}. (c) Error of H_{11}. (d) Magnitude of H_{43}. (e) Angle of H_{43}. (f) Error of H_{43}. (g) Magnitude of H_{62}. (h) Angle of H_{62}. (i) Error of H_{62}.

time of solving the 334 small linear systems and getting the discrete reduced transfer functions is only 0.02 s. In contrast, computing the original transfer function at a single frequency sample needs 136 s so that computing the original transfer function at 334 training samples needs more than 12 h. The time of computing the projection matrix V with an improved version[1] of Algorithm 2 takes 4.5 h. Finally, generating the data of the discrete reduced transfer function is still much faster than generating the data of the original transfer function.

2) Discussion on the Results of DFNN: This part presents the performance of the deep feed-forward neural network. We only show the results using $\ell = 200$ discrete reduced transfer functions as the training data. However, if only the data of the original transfer function are available, then the DFNN can also be straightforwardly applied to those data. Also, in Fig. 4, we show the results of the DFNN method (denoted as NN). There, the magnitudes and angles of the

[1]A multifidelity version of Algorithm 2 is outside of the scope of this work and is presented in a separate paper.

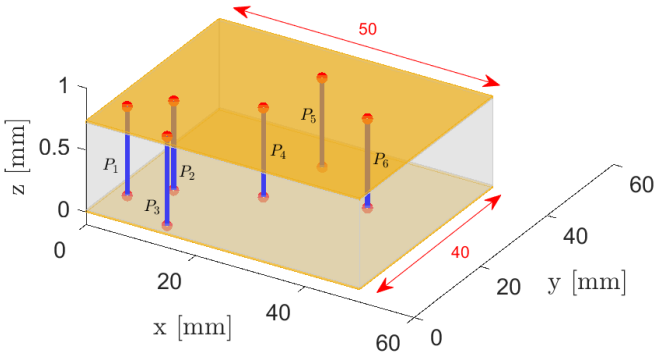


Fig. 5. Power bus geometry. All the dimensions are in mm. The thickness of conductors and dielectrics are 20 μm and 0.7 mm, respectively.

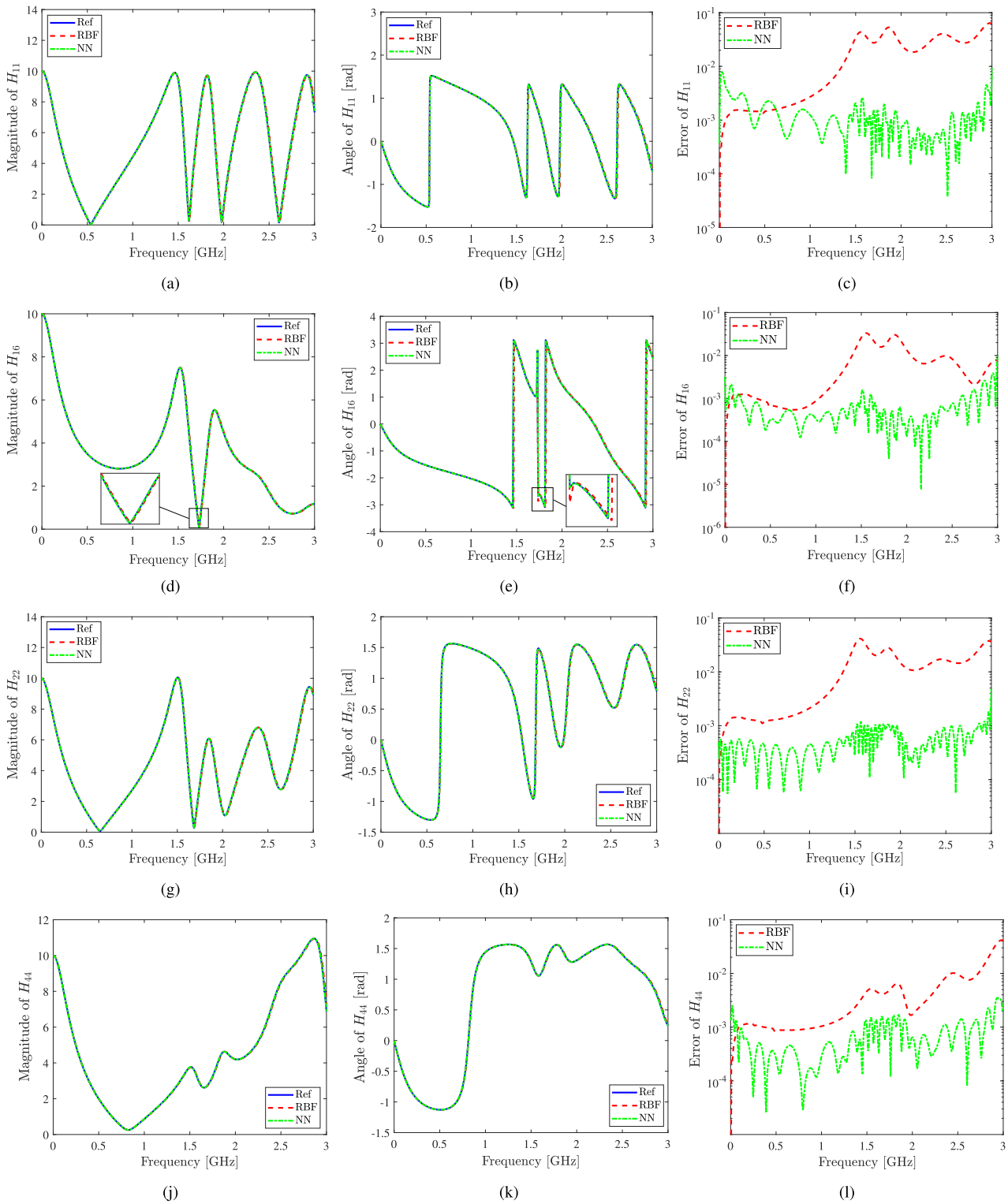


Fig. 6. Experimental results for the powerbus example. (a) Magnitude of H_{11}. (b) Angle of H_{11}. (c) Error of H_{11}. (d) Magnitude of H_{16}. (e) Angle of H_{16}. (f) Error of H_{16}. (g) Magnitude of H_{22}. (h) Angle of H_{22}. (i) Error of H_{22}. (j) Magnitude of H_{44}. (k) Angle of H_{44}. (l) Error of H_{44}.

ROMs H_{11}^{NN}, H_{43}^{NN}, and H_{62}^{NN} are compared with those of the original transfer function. The results are very accurate, and the errors are all below $2 \cdot 10^{-2}$. We use three DFNNs to compute the three ROMs, respectively. The DFNNs are constructed using TensorFlow. In Table II, we show the relevant parameters used in the DFNNs and their runtimes. In both Tables II and IV for the second model and Table VI for the third model, we present the number of hidden layers, the number of neurons at each hidden layer, and the number of epochs during training the network.

TABLE II

COPLANAR MODEL. PERFORMANCE OF THE DFNN WITH THE SINUSOIDAL ACTIVATION FUNCTION tf.math.sin, 4 HIDDEN LAYERS, 20 NEURONS, AND $\ell = 200$

	$\text{Error}_{\max}$	Runtime [s]
H_{11}^{DFNN}	$5 \cdot 10^{-3}$	213
H_{43}^{DFNN}	$1.1 \cdot 10^{-2}$	168
H_{62}^{DFNN}	$1.26 \cdot 10^{-2}$	166

The runtime includes the offline time of training the deep neural network and the online time of ROM prediction over the test samples. The online prediction time for both test examples is below 0.1 s and is almost negligible compared to the training time. Given the same training data, the DFNN-ROMs are more accurate than the RBF-ROMs, and the maximal errors are almost one order of magnitude smaller. It is not surprising that the DFNN training time is also much longer than the RBF training time.

B. Test 2: A Powerbus Model

The second example is a PEEC model of a powerbus illustrated in Fig. 5. The order of the original model is $n = 7576$. It is a model with $n_I = 6$ input ports and $n_O = 6$ output ports.

For this example, no system matrices $E_j, A_j, j = 1, \ldots, d$ in (3) are available. There is only a black-box model of $G(s)$. Algorithm 4 is then used to generate the discrete reduced transfer function at 200 frequency samples, which takes 1.11 h including the time for computing the projection matrix V. If the data of the original transfer function are generated, then it will take 1.78 h to generate the training data at 200 frequency samples. Algorithm 2 is implemented to compute the projection matrix $V \in \mathbb{R}^{n \times r}$ that is then used to compute discrete reduced transfer functions. Algorithm 2 converges in five iterations. A training set Ξ of 30 frequency samples in Algorithm 2 is sufficient to compute a matrix V that can generate accurate discrete reduced transfer functions at any desired frequency sample. To compute the 200 discrete reduced transfer functions, $n_{\text{LS}} = 5 \times 2 = 10$ large-scale systems of dimension $n = 7576$ and $n_{\text{SS}} = 200$ small systems of dimension $r = 56$ need to be solved. If the data of the original transfer function are used to construct the neural network, then 200 large-scale systems must be solved.

From the first example, we see that using the data of the discrete reduced transfer function produces results as accurate as those using the data of the original transfer function. Therefore, we only present the results of RBF interpolation and DFNN using the data of the discrete reduced transfer for this model.

We use four representative entries of the original transfer function to show the performance of the proposed RBF-ROMs and the DFNN-ROMs. ROMs with similar accuracy are obtained for the other entries of the transfer function but are not shown here to avoid redundancy. Both methods use the same training data of $\hat{H}(s_i), i = 1, \ldots, 200$, computed from Algorithm 4. The results of both the RBF-ML approach and the DFNN can be found in Fig. 6.

TABLE III

POWERBUS MODEL. PERFORMANCE OF THE RBF NETWORK (SEE ALGORITHM 6) WITH $\ell = 200$, $n_{\text{LS}} = 10$, AND $n_{\text{SS}} = 200$

$\text{Error}_{\max} H_{11}^{\text{RBF}}$	$6.4 \cdot 10^{-2}$
$\text{Error}_{\max} H_{22}^{\text{RBF}}$	$4.2 \cdot 10^{-2}$
$\text{Error}_{\max} H_{44}^{\text{RBF}}$	$4.2 \cdot 10^{-2}$
$\text{Error}_{\max} H_{16}^{\text{RBF}}$	$3.3 \cdot 10^{-2}$
Runtime [s]	0.8

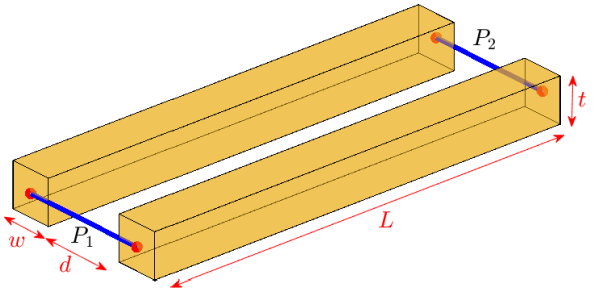


Fig. 7. Transmission line. Geometrical parameters: $L = 50$ cm, $w = 381$ μm, $t = 381$ μm, and $d = 1.524$ mm.

1) Discussion on the Results of RBF Interpolation: In Fig. 6, the largest error of RBF interpolation is $6.4 \cdot 10^{-2}$ for the entry H_{11}. The overall approximation for both the magnitudes and the angles is of acceptable accuracy. In Table III, we present the performance of Algorithm 6. After the training data at $\ell = 200$ training frequency samples are computed, the RBF network is trained, and the trained RBF-ROM is used as the ROM of $H_{ij}(s)$ to compute the approximate transfer function $H_{ij}^{\text{RBF}}(s)$. The runtime in the table includes the training time and the time of computing the approximate transfer function $H_{ij}^{\text{RBF}}(s)$ at 400 testing frequency samples. The runtime for different entries of $H(s)$ is similar, and we show only the runtime for a single entry.

2) Discussion on the Results of DFNN: Fig. 6 also presents the results of DFNN, which are very accurate, and the errors are all below 10^{-2}. Again, DFNN-ROMs have much smaller errors than RBF-ROMs, given the same training data. We use four DFNNs to compute the four ROMs, respectively. Table IV shows the parameters used in the DFNNs and the runtimes of DFNNs. For this model, more hidden layers are needed to get ROMs with acceptable accuracy. H_{11}^{NN} needs more hidden layers and neurons but has a relatively large error; this is probably due to the many sharp tips in the waveform of H_{11}. We also observed that further increasing the number of neurons at each hidden layer does not help to increase the accuracy.

C. Test 3: A Long Transmission Line

The third example is a PEEC model of a long transmission line described in Fig. 7. The order of the original model is $n = 4008$. It is a model with $n_I = 2$ input ports and $n_O = 2$ output ports.

Similar to the second example, no system matrices E_j, $A_j, j = 1, \ldots, d$ in (3) are extracted, and only a black-box

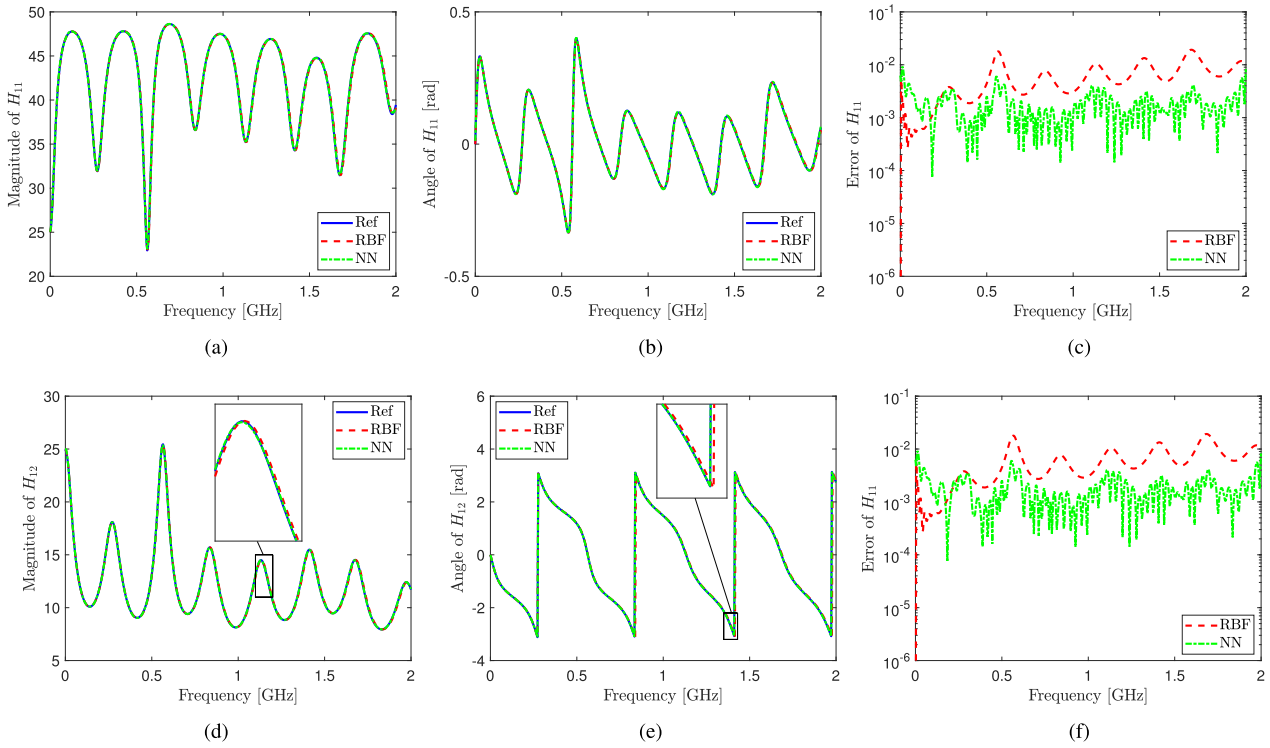


Fig. 8. Experimental results for the long transmission line example. (a) Magnitude of H_{11}. (b) Angle of H_{11}. (c) Error of H_{11}. (d) Magnitude of H_{12}. (e) Angle of H_{12}. (f) Error of H_{12}.

TABLE IV

POWERBUS MODEL. PERFORMANCE OF THE DFNN WITH THE SINUSOIDAL ACTIVATION FUNCTION tf.math.sin AND 20 NEURONS

	hidden layers	epochs	$\text{Error}_{\max}$	Runtime [s]
H_{11}^{DFNN}	10	13000	$9.4 \cdot 10^{-3}$	196
H_{22}^{DFNN}	8	13000	$4.1 \cdot 10^{-3}$	157
H_{44}^{DFNN}	8	13000	$6.1 \cdot 10^{-3}$	161
H_{16}^{DFNN}	6	13000	$8.9 \cdot 10^{-3}$	132

model of $G(s)$ is available. We aim at approximating the original transfer function at 500 frequency samples. To this end, the discrete reduced transfer functions are computed every three sample points, resulting in a group of training data at 167 frequency samples. We use Algorithm 4 to generate the discrete reduced transfer function, which takes 15 min including the time of computing the projection matrix V. If the training data of the original transfer function are generated, then it will take 19 min. Algorithm 2 converges in 13 iterations. A training set Ξ of 30 frequency samples in Algorithm 2 is used to compute the matrix V that constructs accurate discrete reduced transfer functions at any frequency samples.

To compute the 167 discrete reduced transfer functions, $n_{\text{LS}} = 13 \times 2 = 26$ large-scale systems of dimension $n = 4008$ and $n_{\text{SS}} = 167$ small systems of dimension $r = 52$ need to be solved. If the data of the original transfer function are used to construct the neural network, then 167 large-scale systems have to be solved.

TABLE V

TRANSMISSION LINE MODEL. PERFORMANCE OF THE RBF NETWORK (SEE ALGORITHM 6) WITH $\ell = 167$, $n_{\text{LS}} = 26$, AND $n_{\text{SS}} = 167$

$\text{Error}_{\max} H_{11}^{\text{RBF}}$	$1.9 \cdot 10^{-2}$
$\text{Error}_{\max} H_{12}^{\text{RBF}}$	$3.6 \cdot 10^{-2}$
Runtime [s]	0.8

The same training data of $\hat{H}(s_i), i = 1, \ldots, 167$ are used to train both the RBF network and the DFNN. The results of both methods for predicting the original transfer function at all the 500 frequency samples can be found in Fig. 8. Since the system is linear, it turns out to be reciprocal, and therefore, $H_{12}(s) = H_{21}(s)$. Furthermore, due to the symmetry of the system with respect to the ports, it also holds that $H_{11}(s) = H_{22}(s)$. For these reasons, only the results for $H_{11}(s)$ and $H_{12}(s)$ are presented.

1) Discussion on the Results of RBF Interpolation: In Fig. 8, the largest error of RBF interpolation is $3.6 \cdot 10^{-2}$ for the entry H_{12}. The overall approximation is sufficiently accurate. In Table V, we present the performance of Algorithm 6. The RBF network is trained using 167 training data. The trained RBF network is used as the ROM of $H_{ij}(s)$ to compute the approximate transfer function $H_{ij}^{\text{RBF}}(s)$. The runtime in the table includes the training time and the prediction time: time of computing the approximate transfer function $H_{ij}^{\text{RBF}}(s)$ at 500 frequency samples. The runtime for approximating different entries of $H(s)$ is similar, and we show only the runtime for computing the entry $H_{11}^{\text{RBF}}(s)$.

TABLE VI
TRANSMISSION LINE MODEL. PERFORMANCE OF THE DFNN WITH THE SINUSOIDAL ACTIVATION FUNCTION tf.math.sin, 6 HIDDEN LAYERS, AND 20 NEURONS

	epochs	$\text{Error}_{\max}$	Runtime [s]
H_{11}^{DFNN}	20000	$9.2 \cdot 10^{-3}$	215
H_{12}^{DFNN}	14000	$1.74 \cdot 10^{-2}$	150

2) Discussion on the Results of DFNN: Fig. 8 also presents the results of DFNN, which are again more accurate than the RBF prediction. The parameters used in the DFNNs, the errors of the DFNN prediction, and the runtimes of DFNNs are presented in Table VI. Six hidden layers are able to get accurate ROMs. To predict H_{12} or H_{21}, we use fewer epochs while producing larger errors. However, we also used a larger number of epochs, such as 20000, but with only slightly improved accuracy. The resulting maximal error is $1.74 \cdot 10^{-2}$. We have noticed that the maximal error appears at a lower frequency, where not enough samples were taken. Taking more samples at the low-frequency range would reduce the error. Since the overall error is already small enough, there is no need to further improve the accuracy.

For all three models, we use the sinusoidal activation function tf.math.sin in Tensorflow. Other activation functions cannot construct ROMs as accurately as tf.math.sin. It is worth pointing out that the waveform plotted using the training data of the discrete transfer functions is already close to the waveform of the original transfer function, which can help us to choose the proper activation functions. Except for a few, most of the waveforms of the transfer functions are closer to the sinusoidal function and could be better predicted using the sinusoidal activation function.

V. CONCLUSION

We have proposed data-driven MOR methods using ML. ROMs obtained from both RBF networks and deep forward neural networks can accurately predict the original transfer function at testing frequencies. The RBF networks need a much shorter time to be trained but are generally less accurate than the DFNNs. The proposed methods will be tested with more challenging examples in the future. Other DL networks, e.g., convolutional neural networks, will be investigated to further improve the efficiency of the proposed methods. The extension of the proposed method to time domain analysis, along with passivity and causality properties, will also be investigated in the future.

REFERENCES

[1] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM Rev.*, vol. 57, no. 4, pp. 483–531, 2019.

[2] A. Quarteroni and G. Rozza, *Reduced Order Methods for Modeling and Computational Reduction* (Modeling, Simulationand Applications), vol. 9. Cham, Switzerland: Springer, 2014.

[3] A. C. Antoulas, C. A. Beattie, and S. Gugercin, *Interpolatory Methods for Model Reduction* (Computational Science & Engineering). Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2020.

[4] P. Benner, S. Grivet-Talocia, A. Quarteroni, G. Rozza, W. Schilders, and L. M. Silveira, *Model Order Reduction*, vol. 1. Berlin, Germany: De Gruyter, 2021.

[5] A. Charest, D. Saraswat, M. Nakhla, R. Achar, and N. Soveiko, "Compact macromodeling of high-speed circuits via delayed rational functions," *IEEE Microw. Wireless Compon. Lett.*, vol. 17, no. 12, pp. 828–830, Dec. 2007.

[6] A. Chinea, P. Triverio, and S. Grivet-Talocia, "Delay-based macromodeling of long interconnects from frequency-domain terminal responses," *IEEE Trans. Adv. Packag.*, vol. 33, no. 1, pp. 246–256, Feb. 2010.

[7] C. A. Beattie and S. Gugercin, "Model reduction by rational interpolation," in *Model Reduction Approximation: Theory Algorithms*. Philadelphia, PA, USA: SIAM, 2017.

[8] A. Zanco and S. Grivet-Talocia, "Toward fully automated high-dimensional parameterized macromodeling," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 11, no. 9, pp. 1402–1416, Sep. 2021.

[9] B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting," *IEEE Trans. Power Del.*, vol. 14, no. 3, pp. 1052–1061, Jul. 1999.

[10] S. Grivet-Talocia and B. Gustavsen, *Passive Macromodeling: Theory and Applications* (Microwave and optical engineering). Hoboken, NJ, USA: Wiley, 2015.

[11] A. C. Ionita and A. C. Antoulas, "Data-driven parametrized model reduction in the Loewner framework," *SIAM J. Sci. Comput.*, vol. 36, no. 3, pp. A984–A1007, Jan. 2014.

[12] Y. Nakatsukasa, O. Sète, and L. N. Trefethen, "The AAA algorithm for rational approximation," *SIAM J. Sci. Comput.*, vol. 40, no. 3, pp. 1494–1522, Jan. 2018.

[13] Y. Q. Xiao, S. Grivet-Talocia, P. Manfredi, and R. Khazaka, "A novel framework for parametric Loewner matrix interpolation," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 9, no. 12, pp. 2404–2417, Dec. 2019.

[14] P. Schulze and B. Unger, "Data-driven interpolation of dynamical systems with delay," *Syst. Control Lett.*, vol. 97, pp. 125–131, Nov. 2016.

[15] P. Schulze, B. Unger, C. Beattie, and S. Gugercin, "Data-driven structured realization," *Linear Algebra Appl.*, vol. 537, pp. 250–286, Jan. 2018.

[16] S. Fresca, A. Manzoni, L. Dede, and A. Quarteroni, "Deep learning-based reduced order models in cardiac electrophysiology," *PLoS ONE*, vol. 15, no. 10, pp. 1–32, 2020.

[17] F. Regazzoni, L. Dedè, and A. Quarteroni, "Machine learning of multiscale active force generation models for the efficient simulation of cardiac electromechanics," *Comput. Methods Appl. Mech. Eng.*, vol. 370, Oct. 2020, Art. no. 113268.

[18] S. M. Rahman, S. Pawar, O. San, A. Rasheed, and T. Iliescu, "Nonintrusive reduced order modeling framework for quasigeostrophic turbulence," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 100, no. 5, Nov. 2019, Art. no. 053306.

[19] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart, "Model reduction and neural networks for parametric PDEs," *SMAI J. Comput. Math.*, vol. 7, pp. 121–157, Jul. 2021.

[20] S. A. Renganathan, R. Maulik, and V. Rao, "Machine learning for non-intrusive model order reduction of the parametric inviscid transonic flow past an airfoil," *Phys. Fluids*, vol. 32, no. 4, Apr. 2020, Art. no. 047110.

[21] S. Dutta, M. W. Farthing, E. Perracchione, G. Savant, and M. Putti, "A greedy non-intrusive reduced order model for shallow water equations," *J. Comput. Phys.*, vol. 439, Aug. 2021, Art. no. 110378.

[22] M. Kast, M. Guo, and J. S. Hesthaven, "A non-intrusive multi-delity method for the reduced order modeling of nonlinear problems," *Comp. Meth. Appl. Mech. Eng.*, vol. 364, p. 112947, Jun. 2020.

[23] A. Berzins, J. Helmig, F. Key, and S. Elgeti, "Standardized nonintrusive reduced order modeling using different regression models with application to complex flow problems," 2020, *arXiv:2006.13706*.

[24] W. Chen, Q. Wang, J. S. Hesthaven, and C. Zhang, "Physics-informed machine learning for reduced-order modeling of nonlinear problems," *J. Comput. Phys.*, vol. 446, Dec. 2021, Art. no. 110666.

[25] S. Fresca, L. Dede, and A. Manzoni, "A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs," *J. Sci. Comput.*, vol. 87, no. 2, p. 61, May 2021.

[26] L. Feng and P. Benner, "Model order reduction for systems with non-rational transfer function arising in computational electromagnetics," in *Scientific Computing in Electrical Engineering (SCEE)* (Mathematics in Industry), vol. 14, J. Roos and L. R. J. Costa, Eds. Berlin, Germany: Springer-Verlag, 2010, pp. 512–522.

[27] L. Feng and P. Benner, "On error estimation for reduced-order modeling of linear non-parametric and parametric systems," *ESAIM, Math. Model. Numer. Anal.*, vol. 55, no. 2, pp. 561–594, Mar. 2021.
[28] H. Wendland. *Scattered Data Approximation* (Cambridge Monographs on Applied and Computational Mathematics), vol. 17. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[29] M. A. Mongillo. *Choosing Basis Functions and Shape Parameters for Radial Basis Function Methods*, vol. 4. Philadelphia, PA, USA: SIAM Undergraduate Research Online (SIUO), Dec. 2011, pp. 190–209.

Lihong Feng received the Ph.D. degree in computational mathematics from Fudan University, Shanghai, China, in 2002.

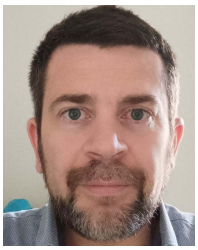
She worked at the host university Technische Universität Chemnitz (TU Chemnitz), Chemnitz, Germany. Since 2010, she has been a Senior Scientist with the Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany, where she became the Team Leader of the CSC Group headed by Peter Benner. Her research interests include model-order reduction and fast simulation of complex models arising from engineering applications, such as fluid dynamics, chemical engineering, mechanical engineering, electrical engineering, numerical analysis, and scientific computing.

Dr. Feng received an Alexander-von-Humboldt Fellowship for the period 2007–2009.

Peter Benner received the Diploma degree in mathematics from RWTH Aachen University, Aachen, Germany, in 1993, the Ph.D. degree in mathematics from the University of Kansas, Lawrence, KS, USA, and the Technische Universität Chemnitz (TU Chemnitz), Zwickau, Germany, in February 1997, and the Habilitation (Venia Legendi) degree in mathematics from the University of Bremen, Bremen, Germany, in 2001.

After spending a term as a Visiting Associate Professor at the Hamburg University of Technology (TU Hamburg), Hamburg, Germany, he was a Lecturer in mathematics with the Berlin Institute of Technology (TU Berlin), Berlin, Germany, from 2001 to 2003. Since 2003, he has been a Professor of mathematics in industry and technology with TU Chemnitz. In 2010, he was appointed as one of the four directors of the Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany. Since 2011, he has been an Honorary Professor with the Otto-von-Guericke University of Magdeburg, Magdeburg. His research interests include scientific machine learning and computing, numerical mathematics, systems theory, and optimal control.

Dr. Benner is a SIAM Fellow (Class of 2017).



Daniele Romano was born in Campobasso, Italy, in 1984. He received the Laurea degree in computer science and automation engineering from the University of L'Aquila, L'Aquila, Italy, in 2012, and the Ph.D. degree from the Università degli Studi dell'Aquila, in 2018.

Since 2012, he has been with the UAq electromagnetic compatibility (EMC) Laboratory, University of L'Aquila, focusing on EMC modeling and analysis, algorithm engineering, and speed-up techniques applied to EMC problems.

Giulio Antonini (Senior Member, IEEE) received the Laurea degree (cum laude) in electrical engineering from the University of L'Aquila, L'Aquila, Italy, in 1994, and the Ph.D. degree in electrical engineering from the University of Rome "La Sapienza," Rome, Italy, in 1998.

Since 1998, he has been with the UAq electromagnetic compatibility (EMC) Laboratory, University of L'Aquila, where he is currently a Professor. He has authored more than 300 papers published in international journals and in the proceedings of international conferences. He has also coauthored the book *Circuit Oriented Electromagnetic Modeling Using the PEEC Techniques* (Wiley–IEEE Press, 2017). His scientific interests are in the field of computational electromagnetics.