

UC Berkeley

UC Berkeley Previously Published Works

Title

NEURAL EMPIRICAL INTERPOLATION METHOD FOR NONLINEAR MODEL REDUCTION

Permalink

<https://escholarship.org/uc/item/1dx2s00k>

Journal

SIAM JOURNAL ON SCIENTIFIC COMPUTING, 47(6)

ISSN

1064-8275

Authors

Hirsch, Max

Pichi, Federico

Hesthaven, Jan S

Publication Date

2025

DOI

10.1137/24M1681434

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed

NEURAL EMPIRICAL INTERPOLATION METHOD FOR NONLINEAR MODEL REDUCTION

MAX HIRSCH, FEDERICO PICHI, AND JAN S. HESTHAVEN

ABSTRACT. In this paper, we introduce the neural empirical interpolation method (NEIM), a neural network-based alternative to the discrete empirical interpolation method for reducing the time complexity of computing the nonlinear term in a reduced order model (ROM) for a parameterized nonlinear partial differential equation. NEIM is a greedy algorithm which accomplishes this reduction by approximating an affine decomposition of the nonlinear term of the ROM, where the vector terms of the expansion are given by neural networks depending on the ROM solution, and the coefficients are given by an interpolation of some “optimal” coefficients. Because NEIM is based on a greedy strategy, we are able to provide a basic error analysis to investigate its performance. NEIM has the advantages of being easy to implement in models with automatic differentiation, of being a nonlinear projection of the ROM nonlinearity, of being efficient for both nonlocal and local nonlinearities, and of relying solely on data and not the explicit form of the ROM nonlinearity. We demonstrate the effectiveness of the methodology on solution-dependent and solution-independent nonlinearities, a nonlinear elliptic problem, and a nonlinear parabolic model of liquid crystals.

Code availability: <https://github.com/maxhirsch/NEIM>

1. INTRODUCTION

In the numerical solution of Partial Differential Equations (PDEs) [36], it is often necessary to study the behavior of the physics-based model corresponding to many different parameter configurations. This is the case, for example, in inverse problems, design, and uncertainty quantification, where physical or geometric properties of the problem need to be investigated. Approximating the solutions of the PDE via high-fidelity simulations usually involves solving a system of equations of large dimensionality, and obtaining many-query and real-time simulations becomes prohibitively expensive. Reduced order modeling (ROM) [5, 6] consists of a wide class of approaches in which such solutions are approximated via surrogate models with lower dimensional systems of equations. Because of this reduced or latent representation, the system can be solved more efficiently, allowing for a reduction of the computational cost. ROM techniques are divided into an offline stage and an online stage, where the information collected via high fidelity solvers during the former is exploited for fast online evaluations. This is possible thanks to the *reduced basis* space, a low-dimensional space spanned by an orthonormal basis which can be obtained via well-known approaches such as the Greedy method or the Proper Orthogonal Decomposition (POD) [21, 35]. Within the POD-based model order reduction framework, the high fidelity or full-order approximation of the solution is computed and stored offline for several parameter instances, forming the so-called *snapshots*. Then, a singular value decomposition of the snapshots matrix is performed to retain the hopefully few left singular vectors capturing most of the system’s behavior. A Galerkin projection is then performed on the subspace encoded by the extracted reduced basis, and the high fidelity problem is reformulated in a lower dimensional space. In the online stage, the lower dimensional problem can be solved efficiently and accurately for many different parameter configurations.

To obtain such computational speedup, two main assumptions need to be satisfied: (i) the problem is reducible, meaning that we can compress the information with a small number of basis functions, i.e., its Kolmogorov n -width has a fast decay, and (ii) we can efficiently assemble the parameter-dependent quantities thanks to the affine-decomposition property, allowing us to have an online stage that does not depend on the potentially high number of degrees of freedom of the problem. The former is related to the

(MH) DEPARTMENT OF MATHEMATICS, UNIVERSITY OF CALIFORNIA BERKELEY, BERKELEY, CA 94720, USA.

(FP) MATHLAB, MATHEMATICS AREA, SISSA, VIA BONOMEA 265, I-34136 TRIESTE, ITALY.

(JSH, FP) MCSS, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 1015 LAUSANNE, SWITZERLAND.

E-mail addresses: mhirsch@berkeley.edu, fpichi@sissa.it, jan.hesthaven@epfl.ch.

physical complexity of the model, while the latter can be satisfied choosing ad-hoc strategies to restore the computational savings.

Indeed, when dealing with linear PDEs with affine parameter dependence, the POD-Galerkin approach results in a linear system of equations of dimension much smaller than the original problem, leading to consistent speedups. However, when the PDE is nonlinear, POD-Galerkin will not in general result in significant speedups because the time complexity of evaluating the nonlinearity in the reduced order system of equations still depends on the dimensionality of the high fidelity problem.

For this reason, *hyper-reduction* techniques have been developed to satisfy the assumption (ii) above, improving the time complexity of evaluating nonlinearities thanks to a further level of model reduction. Among the most common hyper-reduction techniques, there are the Empirical Interpolation Methods (EIM) [3] and its variant the Discrete Empirical Interpolation Methods [9]. In DEIM, one performs an oblique projection onto a subspace of the column space of nonlinearities computed in the offline phase. This projection is chosen in such a way that only a smaller number of entries of the nonlinearity need to be computed in the online phase, and for this reason, it allows for computational cost reduction when dealing with componentwise nonlinearities. For nonlocal nonlinearities, in which the components of the nonlinearity depend on every input, e.g. integral kernels or other integral operators like the Boltzmann collision operator [15], exploiting DEIM would not be effective. Moreover, it can only be applied if one has access to the exact form of the nonlinearity in the system, and if this nonlinearity is not programmed to be compatible with automatic differentiation [4], it is not easily used in workflows requiring deep learning. This is the case, for example, in existing reduced order modeling libraries like RBniCS [39].

In this paper, we develop a data-based hyper-reduction machine-learning technique for ROM, compatible with automatic differentiation, which we call the Neural Empirical Interpolation Method (NEIM). The idea of NEIM is to approximate an affine decomposition of the nonlinearity in a reduced order model so that the nonlinear term can be written as the sum of the product between parameter-dependent coefficients and solution-dependent neural networks. This approximation is done with a greedy approach. Given a fixed set of parameters, NEIM first finds the parameter corresponding to the reduced order nonlinearity with highest approximate L^2 error. A neural network is then trained to approximate a normalized version of this nonlinearity, and the coefficient of the first term in the affine decomposition is determined by solving a linear system of equations. This is repeated iteratively, with the L^2 error computed relative to the current approximation of the affine decomposition. At the end of this iterative procedure, the approximations for the coefficients in the affine decomposition are interpolated from the fixed set of parameters to the whole parameter space. Because of the greedy strategy, we are able to perform a basic error analysis to investigate the performance of NEIM. Namely, we can show that the error estimate used in the algorithm is decreasing with each iteration. We also show how the final generalization error is composed of projection, training, and interpolation errors. NEIM has the advantage over DEIM of being a completely data-driven approach, which gives NEIM the main advantage of being a far less intrusive method and more easily integrated with other deep learning techniques. Indeed, in the online phase of solving a reduced order model with NEIM, the nonlinear terms in the original full order model are never evaluated, unlike DEIM which still evaluates the nonlinearity at selected rows. Thus, the full order model code does not need to be changed to exploit the NEIM strategy. Moreover, unlike DEIM which involves a linear projection and deals with componentwise nonlinearities, NEIM is a nonlinear methodology which is also suitable for the previously described nonlocal nonlinearities.

We demonstrate the effectiveness of our methodology on solution-independent and solution-dependent nonlinearities, a nonlinear elliptic problem, and a nonlinear parabolic model of liquid crystals. In the solution-independent test case, we use NEIM to approximate a nonlinearity which depends only on the PDE parameter μ and not the PDE solution. In this case, the neural networks in the approximation should be approximately constant. In the solution-dependent test case, we modify the nonlinearity in the first example so that the neural networks should learn non-constant functions. Each of these first two methods are based on a finite difference method. The third example presents an application of NEIM to a nonlinear elliptic problem which is solved with a physics-informed neural network (PINNs) [37] in the ROM context [10]. We then present an application of the methodology to a system of PDEs describing the dynamics of liquid crystal molecules which is discretized with finite elements and has two nonlinearities to approximate. In addition to the advantages of the proposed methodology, NEIM can easily be used in combination with automatic differentiation in deep learning libraries like PyTorch [32]. This is unlike DEIM in finite element

settings in which the assembly of the nonlinearity is not easy to interface between deep learning libraries and finite element libraries. We show the compatibility of NEIM with automatic differentiation in our example with a physics-informed neural network. Furthermore, without relying on a complete approximation of the nonlinear field, NEIM provides more control and interpretability, also thanks to the developed error analysis. Lastly, these architectures are by definition small and fast to train and evaluate, allowing for their usage in standard reduction settings.

There exist several machine learning approaches to model order reduction for parameterized PDEs which tackle the hyper-reduction issue from a different perspective. These can be classified into linear and nonlinear approaches, depending on the maps exploited to approximate the solution field. As an example, neural networks [22, 33, 10], interpolation [8, 13], and Gaussian process regression [20, 23] have been used to map PDE parameters to coefficients of a linear basis for a reduced order subspace. The focus of such works, however, is on approximating the entire parameter-solution mapping corresponding to the reduced order problem and not on designing a hyper-reduction technique to be used online. On the other hand, nonlinear compression has gained much interest to deal with, e.g., advection-dominated problems [14, 41, 42, 24], and more generally to overcome the Kolmogorov barrier [31] by exploiting autoencoder architectures [26, 38, 17, 34, 30]. Since the complex parameter-solution mapping in the nonlinear setting is usually obtained through deep neural networks, the approximated map can be subject to overfitting, require many data to be trained, and lack interpretability. The NEIM approach is also closely related to very recent works exploring the usage of empirical interpolation techniques in combination with neural networks [2, 11], and to the development of adaptive methods of constructing suitable basis functions [1, 16, 18, 12]. Lastly, there are more general machine learning approaches to approximating nonlinear operators in scientific computing [25, 7]. In particular, DeepONets are a popular method used for this purpose [28]. In the most basic setting, DeepONets can be thought of as having one neural network which takes in a parameter and another network which takes the evaluations of a function at various points in space. The output of the DeepONet is the dot product of the outputs of these two networks. From this perspective, NEIM can be thought of structurally as a specific instance of DeepONet. The difference between NEIM approximations and DeepONets is the way in which NEIM is fit to data in the offline phase. NEIM relies on a greedy method of selecting parameters corresponding to individual terms in the approximation sum. In effect, it is a greedy method of producing a sequence of progressively more accurate DeepONets.

The remainder of the paper is organized as follows. Section 2 presents the ROM framework which motivates NEIM. Section 3 defines the NEIM algorithm, and Section 4 presents numerical results. Finally, Section 5 discusses conclusions.

2. PROBLEM SETTING

We begin by describing our setting of reduced order models. Let $p, d \in \mathbb{N}$, and consider $\mathcal{P} \subset \mathbb{R}^p$ a compact parameter space and $\Omega(\mu) \subset \mathbb{R}^d$ a bounded spatial domain, where $\mu \in \mathcal{P}$. Then for $\mu \in \mathcal{P}$, consider the nonlinear PDE

$$(1) \quad \begin{aligned} \mathcal{A}(\mu)[v(x)] + \mathcal{N}(\mu)[v(x)] &= f(\mu), & x \in \Omega(\mu) \\ \mathcal{B}(\mu)[v(x)] &= 0, & x \in \partial\Omega(\mu), \end{aligned}$$

where $\mathcal{A}(\mu)$ is a linear differential operator, $\mathcal{N}(\mu)$ is a nonlinear differential operator, and $f(\mu)$ is a forcing term. $\mathcal{B}(\mu)$ is a boundary operator defined on the boundary $\partial\Omega(\mu)$ of the spatial domain. We seek a solution $v \in X$, where X is some suitable function space. To solve this problem numerically, we introduce a finite dimensional subspace $X_h \subseteq X$ in which we seek a solution to a weak formulation of Equation (1). Using, for example, a finite element discretization of the above problem, we obtain the nonlinear system of equations (with a slight abuse of notation)

$$(2) \quad A(\mu)v + N(v; \mu) = f(\mu), \quad \mu \in \mathcal{P},$$

where $A(\mu) \in \mathbb{R}^{n \times n}$ is the linear operator, $N(\cdot; \mu) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the nonlinearity, $f(\mu) \in \mathbb{R}^n$ is the forcing term, and $v \in \mathbb{R}^n$.

2.1. Proper Orthogonal Decomposition. Now we describe how we perform the proper orthogonal decomposition (POD) to obtain a reduced order modeling corresponding to the discrete problem in Equation (2). Suppose that we have some number ν of parameters $\mu_1^{POD}, \mu_2^{POD}, \dots, \mu_\nu^{POD}$. To each parameter μ_i^{POD} , there corresponds a solution $v(\mu_i^{POD})$ of (2). We collect these solutions into the snapshot matrix

$S = [v(\mu_1^{POD}) \ \dots \ v(\mu_\nu^{POD})]$. The POD then consists in taking the singular value decomposition of $S = U\Sigma V^T$, where $U \in \mathbb{R}^{n \times n}$, $\Sigma \in \mathbb{R}^{n \times \nu}$, and $V \in \mathbb{R}^{\nu \times \nu}$. We take U_r to be the matrix consisting of the first r columns of U , and approximate the solution of Equation (2) by solving the ROM problem

$$(3) \quad U_r^\top A(\mu) U_r \tilde{v} + U_r^\top N(U_r \tilde{v}; \mu) = U_r^\top f(\mu)$$

so that the solution $v(\mu)$ to Equation (2) and the solution $\tilde{v}(\mu)$ to Equation (3) satisfy $v(\mu) \approx U_r \tilde{v}(\mu)$. Assuming that $A(\mu)$ and $f(\mu)$ admit affine decompositions such that

$$A(\mu) = \sum_{i=1}^{q_1} \theta_i^A(\mu) A_i, \quad f(\mu) = \sum_{i=1}^{q_2} \theta_i^f(\mu) f_i$$

for some coefficients $\theta_i^A(\mu)$ and $\theta_i^f(\mu)$, constants $A_i \in \mathbb{R}^{n \times n}$ and $f_i \in \mathbb{R}^n$, and $q_1, q_2 \in \mathbb{N}$, we see that the time complexity of evaluating $U_r^\top A(\mu) U_r$ and $U_r^\top f(\mu)$ is independent of n and only dependent on r since $U_r^\top A_i U_r$ and $U_r^\top f_i$ can be pre-computed. However, the time complexity of computing $U_r^\top N(U_r \tilde{v}; \mu)$ still depends on n , so we do not see significant improvements in compute time by solving Equation (3) instead of Equation (2). A typical approach to this problem is to use the discrete empirical interpolation method [9]. Here, we instead introduce an iterative method we call the neural empirical interpolation method in which we approximate an affine decomposition of this nonlinear term using a machine learning approach.

2.2. Discrete Empirical Interpolation Method. We now briefly recall the discrete empirical interpolation method (DEIM), where one approximates

$$N(U_r \tilde{v}; \mu) \approx V_k (P^\top V_k)^{-1} P^\top N(U_r \tilde{v}; \mu),$$

where V_k is the matrix of the k left singular vectors of the matrix of nonlinear snapshots of the full-order system $S_N = [N(v(\mu_1^{POD}); \mu_1^{POD}) \ \dots \ N(v(\mu_\nu^{POD}); \mu_\nu^{POD})]$ and $P = [e_{p_1} \ \dots \ e_{p_k}]$ is a matrix with $e_{p_i} = [0 \ \dots \ 0 \ 1 \ 0 \ \dots \ 0]^\top \in \mathbb{R}^n$ the p_i -th column of the identity matrix for $i = 1, \dots, k$. The indices $p_1, \dots, p_k$ are determined by the greedy Algorithm (1). Following the original DEIM paper [9], the notation $\max$ is MATLAB notation so that $[|\rho|, p_\ell] = \max\{|r|\}$ implies $|\rho| = |r_{p_\ell}| = \max_{i=1, \dots, n} \{|r_i|\}$ with the smallest index taken in case of a tie.

Algorithm 1 DEIM

```

1: procedure DEIM( $\{N(v(\mu_i^{POD}); \mu_i^{POD})\}_{i=1}^\nu \subset \mathbb{R}^n$  linearly independent)
2:    $[|\rho|, p_1] = \max\{|N(v(\mu_1^{POD}); \mu_1^{POD})|\}$ 
3:    $V_k = [N(v(\mu_1^{POD}); \mu_1^{POD})], P = [e_{p_1}]$ 
4:   for  $\ell = 2$  to  $\nu$  do
5:     Solve  $(P^\top V_k) \mathbf{c} = P^\top N(v(\mu_\ell^{POD}); \mu_\ell^{POD})$  for  $\mathbf{c}$ 
6:      $r = N(v(\mu_\ell^{POD}); \mu_\ell^{POD}) - V_k \mathbf{c}$ 
7:      $[|\rho|, p_\ell] = \max\{|r|\}$ 
8:      $V_k \leftarrow [V_k \ N(v(\mu_\ell^{POD}); \mu_\ell^{POD})], P \leftarrow [P \ e_{p_\ell}]$ 
9:   end for
10: end procedure

```

3. NEURAL EMPIRICAL INTERPOLATION METHOD

Given the setup in the previous section, our goal is to develop a DEIM-inspired neural network based methodology, namely the Neural Empirical Interpolation Method (NEIM), to find an approximation of the nonlinear term of the form

$$(4) \quad U_r^\top N(U_r \tilde{v}; \mu) \approx \hat{N}(\tilde{v}; \mu) \doteq \sum_{i=1}^k \theta_i(\mu) M_{\mu^{(i)}}(\tilde{v}),$$

where $M_{\mu^{(i)}} : \mathbb{R}^r \rightarrow \mathbb{R}^r$ is the neural network corresponding to $\mu^{(i)}$ for $i = 1, \dots, k$, and $\theta_i(\mu)$ is determined through interpolation of some “optimal” θ_i . We reinterpret the affine decomposition assumption from the neural network point of view, through a linear combination of parameter dependent coefficients θ_i and

parameter independent networks $M_{\mu^{(i)}}$. Note that $\hat{N}(\tilde{v}; \mu) \in \mathbb{R}^r$, while $N(U_r \tilde{v}; \mu) \in \mathbb{R}^n$. Since we only work with the truncated POD basis, in what follows, we drop the subscript r in U_r .

3.1. NEIM First Step. First, we consider a finite set of parameters $\mathcal{M} = \{\mu_i\}_{i=1}^m \subset \mathcal{P}$ to which correspond the snapshots $\mathcal{S} = \{v_i = v(\mu_i)\}_{i=1}^m \subset \mathbb{R}^n$. We denote their projections onto the POD space by $\tilde{v}_i = U^\top v_i$, $i = 1, \dots, m$. When computing the NEIM approximation in the ROM setting, the quantities $\mathcal{M}$ and $\mathcal{S}$ are already available from the POD approach. Then the first step in NEIM is to initialize our approximation $\hat{N}$. We greedily find new terms in the expansion of $\hat{N}$, so we denote the iterates with a superscript, the 0-th iterate being $\hat{N}^{(0)} \equiv 0$.

Having initialized our approximation, we perform the first training step of NEIM, i.e., we find the first term in the expansion of $\hat{N}$. Because $M_{\mu^{(i)}}(\tilde{v})$ in our expansion is not a function of μ , we train $M_{\mu^{(1)}}$ so that $v \mapsto M_{\mu^{(1)}}(U^\top v)$ approximates the projection of the nonlinear term $U^\top N(\cdot; \mu^{(1)})$ for some $\mu^{(1)}$. We choose $\mu^{(1)}$ so that a measurement of error is maximized:

$$\mu^{(1)} = \arg \max_{\mu \in \mathcal{M}} \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(0)}(v_i; \mu)^2,$$

where $\mathcal{E}^{(0)}(v; \mu) = \|U^\top N(v; \mu) - \hat{N}^{(0)}(U^\top v; \mu)\|_2$. In the above expression, $w_e(\mu_i; \mu) \in [0, \infty)$ are “error” weights which can be interpreted as quadrature weights. In other words, for a fixed $\mu \in \mathcal{M}$, the sum above is an approximation of the L^2 error over $\mathcal{P}$ between $U^\top N(v(\cdot); \mu)$ and its approximation $\hat{N}^{(0)}(U^\top v(\cdot); \mu)$, where $v(\cdot)$ denotes the mapping from a parameter μ to its corresponding high fidelity solution denoted $v(\mu)$. Thus $\mu^{(1)}$ is the parameter such that the corresponding L^2 error approximation is maximized. Note that by choosing the weights appropriately, we can also obtain a more local error estimator, e.g., by choosing $w_e(\mu_i; \mu) = \delta_{\mu_i \mu}$, the Kronecker delta. Having chosen $\mu^{(1)}$, we train $M_{\mu^{(1)}}$ so that $M_{\mu^{(1)}}(\tilde{v}_i) \approx U^\top N(v_i; \mu^{(1)}) / \|U^\top N(v_i; \mu^{(1)})\|_2$ for $v_i \in \mathcal{S}$. The main reason for evaluating the nonlinearity for a fixed parameter $\mu^{(1)}$ while varying the solution v_i comes from viewing the nonlinear term as an operator; thus the NEIM expansion we obtain approximates the interpolation in μ of an operator evaluated at particular parameters. To find this first “mode” of the expansion, we take $M_{\mu^{(1)}}$ to be a simple feedforward neural network trained by minimizing the weighted squared loss

$$(5) \quad \sum_{i=1}^m w_i^{(1)}(\mu_i) \left\| M_{\mu^{(1)}}(\tilde{v}_i) - \frac{U^\top N(v_i; \mu^{(1)})}{\|U^\top N(v_i; \mu^{(1)})\|_2} \right\|_2^2$$

with respect to the network parameters, where $w_i^{(1)}(\mu_i) \in [0, \infty)$ for $i = 1, \dots, m$ determine the relative importance of each sample in the training data for the optimization of the neural network “training” weights.

At this point we need to find the μ -dependent scaling factor for $M_{\mu^{(1)}}$. Thus, we seek some $\theta_1^{(1)}(\mu)$ so that $\theta_1^{(1)}(\mu) M_{\mu^{(1)}}(\tilde{v}_i)$ is “close” to $U^\top N(v_i; \mu)$ for $\mu \in \mathcal{M}$, and $v_i \in \mathcal{S}$. More precisely, we perform an optimization task to find $\theta_1^{(1)}(\mu)$ minimizing

$$(6) \quad \sum_{\mu \in \mathcal{M}} \sum_{i=1}^m w_e(\mu_i; \mu) \|U^\top N(v_i; \mu) - \theta_1^{(1)}(\mu) M_{\mu^{(1)}}(\tilde{v}_i)\|_2^2.$$

The new approximation of the projected nonlinearity is $\hat{N}^{(1)}(\tilde{v}; \mu) = \theta_1^{(1)}(\mu) M_{\mu^{(1)}}(\tilde{v})$, $\tilde{v} \in \mathbb{R}^r$. Note that $\theta_1^{(1)}(\mu)$ is only defined for $\mu \in \mathcal{M}$ because the θ coefficients are interpolated from $\mathcal{M}$ to all of $\mathcal{P}$ at the end of the NEIM approximation procedure. We can find the value of $\theta_1^{(1)}(\mu)$ by setting the partials of (6) equal to 0:

$$\begin{aligned} -2 \sum_{i=1}^m w_e(\mu_i; \mu) \left\langle M_{\mu^{(1)}}(\tilde{v}_i), U^\top N(v_i; \mu) - \theta_1^{(1)}(\mu) M_{\mu^{(1)}}(\tilde{v}_i) \right\rangle &= 0 \\ \implies \theta_1^{(1)}(\mu) &= \frac{\sum_{i=1}^m w_e(\mu_i; \mu) \langle M_{\mu^{(1)}}(\tilde{v}_i), U^\top N(v_i; \mu) \rangle}{\sum_{i=1}^m w_e(\mu_i; \mu) \|M_{\mu^{(1)}}(\tilde{v}_i)\|_2^2}, \quad \mu \in \mathcal{M}. \end{aligned}$$

When all of the weights are equal, having trained $M_{\mu^{(1)}}(\tilde{v}_i)$ so that $M_{\mu^{(1)}}(\tilde{v}_i) \approx \frac{U^\top N(v_i; \mu^{(1)})}{\|U^\top N(v_i; \mu^{(1)})\|_2}$, we expect that $\theta_1^{(1)}(\mu^{(1)}) \approx \frac{1}{m} \sum_{i=1}^m \|U^\top N(v_i; \mu^{(1)})\|_2$. Indeed, taking the weights $w_e \equiv C$ gives

$$\theta_1^{(1)}(\mu^{(1)}) \approx \frac{\sum_{i=1}^m C \left\langle \frac{U^\top N(v_i; \mu^{(1)})}{\|U^\top N(v_i; \mu^{(1)})\|_2}, U^\top N(v_i; \mu^{(1)}) \right\rangle}{\sum_{i=1}^m C \left\| \frac{U^\top N(v_i; \mu^{(1)})}{\|U^\top N(v_i; \mu^{(1)})\|_2} \right\|_2^2} = \frac{1}{m} \sum_{i=1}^m \|U^\top N(v_i; \mu^{(1)})\|_2.$$

Now, the approximation $\hat{N}^{(1)}(\tilde{v}; \mu)$ should be accurate for μ close to $\mu^{(1)}$ and $\tilde{v}$ close to $U^\top v$ with $v \in \mathcal{S}$. The first step of NEIM is summarized in Algorithm (2).

Algorithm 2 NEIM First Step

```

1: procedure NEIM_INIT( $\mathcal{M}, \mathcal{S}$ )
2:    $\hat{N}^{(0)} \leftarrow 0$ 
3:    $\mu^{(1)} \leftarrow \arg \max_{\mu \in \mathcal{M}} \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(0)}(v_i; \mu)^2$ 
4:   Train  $M_{\mu^{(1)}}$  so that  $M_{\mu^{(1)}}(\tilde{v}_i) \approx U^\top N(v_i; \mu^{(1)}) / \|U^\top N(v_i; \mu^{(1)})\|_2$  for all  $v_i \in \mathcal{S}$ 
5:   Find  $\theta_1^{(1)}(\mu)$  such that  $\sum_{\mu \in \mathcal{M}} \sum_{i=1}^m w_e(\mu_i; \mu) \|U^\top N(v_i; \mu) - \theta_1^{(1)}(\mu) M_{\mu^{(1)}}(\tilde{v}_i)\|_2^2 = \min!$ 
6:    $\hat{N}^{(1)}(\tilde{v}; \mu) := \theta_1^{(1)}(\mu) M_{\mu^{(1)}}(\tilde{v}), \tilde{v} \in \mathbb{R}^r, \mu \in \mathcal{M}$ 
7: end procedure

```

3.2. NEIM Update Step. To add a term to the NEIM expansion, there are two main modifications of the first step described above. First, we must be careful to orthogonalize our training data with respect to the previously trained neural networks in the NEIM expansion to avoid instability in the estimates of the coefficients $\theta_i(\mu)$. Second, to solve for the coefficients in the NEIM expansion, we must solve a linear system. Indeed, the first step of NEIM can be viewed as solving a 1×1 linear system for the coefficients. In addition to describing these modifications here, we also generalize the other steps of the NEIM update. At iteration j of the procedure, we have already computed $\hat{N}^{(j-1)}$, and we want to update this approximation by adding a term to the NEIM expansion. We begin, as before, by finding the parameter for which the approximation $\hat{N}^{(j-1)}$ has the worst error:

$$(7) \quad \mu^{(j)} = \arg \max_{\mu \in \mathcal{M} \setminus \{\mu^{(1)}, \dots, \mu^{(j-1)}\}} \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(j-1)}(v_i; \mu)^2,$$

where $\mathcal{E}^{(j-1)}(v; \mu)^2 = \|U^\top N(v; \mu) - \hat{N}^{(j-1)}(U^\top v; \mu)\|_2^2$.

By removing previously selected values for μ from $\mathcal{M}$ in (7), we avoid reselecting the same parameters, due to, e.g., a poorly trained corresponding network, possibly causing stability issues. Moreover, there is no reason why a network with a re-selected parameter would be trained any better than the first network corresponding to that parameter. For example, while it is expected that parameters selected by NEIM will correspond to the smallest approximation errors, the results of the neural network training depend on the weights $w_t^{(j)}$, and some choices of these weights (e.g., choosing $w_t^{(j)}(\mu^{(j)}) = 0$) can lead to results with the error at the selected parameter not decreasing much after training compared to the error at other parameters. Now, having chosen $\mu^{(j)}$, we want to train $M_{\mu^{(j)}}$ so that

$$M_{\mu^{(j)}}(\tilde{v}_i) \approx U^\top N(v_i; \mu^{(j)}) / \|U^\top N(v_i; \mu^{(j)})\|_2, \quad \text{for } v_i \in \mathcal{S}.$$

To avoid training instability, $M_{\mu^{(j)}}$ is normalized and orthogonalized with respect to the previously trained neural networks. We define $y^{(j)}(\mu_i) = U^\top N(v_i; \mu^{(j)})$ for $i = 1, \dots, m$, and let $z^{(j)}(\mu_i)$ be the result of performing Gram-Schmidt on $y^{(j)}(\mu_i)$ with respect to the networks $M_{\mu^{(1)}}(\tilde{v}_i), \dots, M_{\mu^{(j-1)}}(\tilde{v}_i)$. We describe this in Algorithm (3).

Remark 1. Note that the maximum number of terms in the NEIM approximation is r if we assume that the neural networks train well. Indeed, in this case, $\{M_{\mu^{(1)}}(\tilde{v}_i), \dots, M_{\mu^{(r)}}(\tilde{v}_i)\}$ would approximate a basis of $\mathbb{R}^r$ for each $i = 1, \dots, m$, and any additional networks in the NEIM approximation would not greatly reduce the approximation error. This can be compared with DEIM which typically interpolates at many more points

than r (with the number of points bounded by the high fidelity dimension). This difference is due to the fact that DEIM approximates a projection of $\mathbf{N}$ in the high fidelity space which is then multiplied by $U^\top$, but NEIM directly approximates $U^\top \mathbf{N}$ in the reduced order space.

Algorithm 3 Output Data Orthogonalization

```

1: procedure GRAM_SCHMIDT( $y^{(j)}(\mu_i), M_{\mu^{(1)}}(\tilde{v}_i), \dots, M_{\mu^{(j-1)}}(\tilde{v}_i)$ )
2:    $z^{(j)}(\mu_i) \leftarrow y^{(j)}(\mu_i)$ 
3:   for  $k = 1, \dots, j - 1$  do
4:      $z^{(j)}(\mu_i) \leftarrow z^{(j)}(\mu_i) - \frac{\langle z^{(j)}(\mu_i), M_{\mu^{(k)}}(\tilde{v}_i) \rangle}{\|M_{\mu^{(k)}}(\tilde{v}_i)\|_2^2} M_{\mu^{(k)}}(\tilde{v}_i)$ 
5:   end for
6:    $z^{(j)}(\mu_i) \leftarrow z^{(j)}(\mu_i) / \|z^{(j)}(\mu_i)\|_2$ 
7: end procedure

```

We train $M_{\mu^{(j)}}$ with the orthogonalized data by minimizing the weighted squared loss

$$(8) \quad \sum_{i=1}^m w_t^{(j)}(\mu_i) \left\| M_{\mu^{(j)}}(\tilde{v}_i) - z^{(j)}(\mu_i) \right\|_2^2$$

with respect to the network parameters, where the weights $w_t^{(j)}(\mu_i) \in [0, \infty)$ for $i = 1, \dots, m$ have the same meaning as in Equation (5). After training $M_{\mu^{(j)}}$, we find $\theta_1^{(j)}(\mu), \dots, \theta_j^{(j)}(\mu)$ for $\mu \in \mathcal{M}$ such that

$$(9) \quad \sum_{\mu \in \mathcal{M}} \sum_{i=1}^m w_e(\mu_i; \mu) \left\| U^\top \mathbf{N}(v_i; \mu) - \sum_{\ell=1}^j \theta_\ell^{(j)}(\mu) M_{\mu^{(\ell)}}(\tilde{v}_i) \right\|_2^2$$

is minimized. This is equivalent to minimizing for each fixed $\mu \in \mathcal{M}$ the quantity

$$\sum_{i=1}^m w_e(\mu_i; \mu) \left\| U^\top \mathbf{N}(v_i; \mu) - \sum_{\ell=1}^j \theta_\ell^{(j)}(\mu) M_{\mu^{(\ell)}}(\tilde{v}_i) \right\|_2^2.$$

This problem amounts to solving the $j \times j$ linear system resulting from setting the partial derivatives of this expression equal to zero. Doing so for $\mu \in \mathcal{M}$, we obtain

$$-2 \sum_{i=1}^m w_e(\mu_i; \mu) \left\langle M_{\mu^{(k)}}(\tilde{v}_i), U^\top \mathbf{N}(v_i; \mu) - \sum_{\ell=1}^j \theta_\ell^{(j)}(\mu) M_{\mu^{(\ell)}}(\tilde{v}_i) \right\rangle = 0, \quad k = 1, \dots, j,$$

which results in the system $\mathbf{A}\theta = \mathbf{b}$ for $\theta = [\theta_1^{(j)}(\mu), \dots, \theta_j^{(j)}(\mu)]^\top$, with

$$(10) \quad \begin{aligned} \mathbf{A}_{k\ell} &= \sum_{i=1}^m w_e(\mu_i; \mu) \langle M_{\mu^{(k)}}(\tilde{v}_i), M_{\mu^{(\ell)}}(\tilde{v}_i) \rangle, \\ \mathbf{b}_k &= \sum_{i=1}^m w_e(\mu_i; \mu) \langle M_{\mu^{(k)}}(\tilde{v}_i), U^\top \mathbf{N}(v_i; \mu) \rangle, \quad k, \ell = 1, \dots, j. \end{aligned}$$

We highlight the fact that the optimization problem defined by the quantity in (9) depends quadratically on m and involves solving m linear systems of size $j \times j$. After solving this system for each $\mu \in \mathcal{M}$, we define the j -th approximation as

$$\hat{\mathbf{N}}^{(j)}(\tilde{v}; \mu) = \sum_{\ell=1}^j \theta_\ell^{(j)}(\mu) M_{\mu^{(\ell)}}(\tilde{v}) \quad \text{for } \tilde{v} \in \mathbb{R}^r, \mu \in \mathcal{M}.$$

When the algorithm meets the termination condition given by a suitable stopping criteria (e.g., the error quadrature (7) is sufficiently small), we either train a neural network or use interpolation to predict the vector of θ_ℓ for a newly given parameter μ . The method can be chosen depending on the preferred regression or interpolation context. The full NEIM training procedure is described in Algorithm (4). We postpone the discussion regarding weights and stopping criteria to the next section.

Algorithm 4 NEIM Training

```

1: procedure NEIM_TRAIN( $\mathcal{M}, \mathcal{S}$ )
2:    $\hat{\mathbf{N}}^{(1)}, \mu^{(1)} \leftarrow \text{NEIM\_INIT}(\mathcal{M}, \mathcal{S})$ 
3:    $j \leftarrow 2$ 
4:   while stopping criteria not met do
5:      $\mu^{(j)} \leftarrow \arg \max_{\mu \in \mathcal{M} \setminus \{\mu^{(1)}, \dots, \mu^{(j-1)}\}} \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(j-1)}(v_i; \mu)^2$ 
6:     for  $i = 1, \dots, m$  do
7:        $y^{(j)}(\mu_i) \leftarrow U^\top \mathbf{N}(v_i; \mu^{(j)})$ 
8:        $z^{(j)}(\mu_i) \leftarrow \text{GRAM\_SCHMIDT}(y^{(j)}(\mu_i), M_{\mu^{(1)}}(\tilde{v}_i), \dots, M_{\mu^{(j-1)}}(\tilde{v}_i))$ 
9:     end for
10:    Train  $M_{\mu^{(j)}}$  by minimizing  $\sum_{i=1}^m w_t^{(j)}(\mu_i) \|M_{\mu^{(j)}}(U^\top v(\mu_i)) - z^{(j)}(\mu_i)\|^2$ 
11:    Define  $\mathbf{A}(\mu), \mathbf{b}(\mu)$  as in (10)
12:     $\begin{bmatrix} \theta_1^{(j)}(\mu) & \dots & \theta_j^{(j)}(\mu) \end{bmatrix}^\top \leftarrow \mathbf{A}(\mu)^{-1} \mathbf{b}(\mu)$  for  $\mu \in \mathcal{M}$ 
13:     $\hat{\mathbf{N}}^{(j)}(\tilde{v}; \mu) := \sum_{\ell=1}^j \theta_\ell^{(j)}(\mu) M_{\mu^{(\ell)}}(\tilde{v})$  for  $\tilde{v} \in \mathbb{R}^r, \mu \in \mathcal{M}$ 
14:     $j \leftarrow j + 1$ 
15:  end while
16:  Perform regression/interpolation for  $\theta_\ell$ 
17: end procedure

```

Remark 2. In the NEIM algorithm, we exploit the weighting of the least squares problems in two different steps. Indeed, we weight the training data in the neural network loss at each iteration (Equation (8)), and we also weight the least squares problem for finding the $\theta(\mu)$ coefficients (Equation (9)). Of these, the weighting of the training data is most important practically speaking. To obtain good results, it is needed that the trained neural networks $M_{\mu^{(j)}}$ are accurate when evaluated at $\tilde{v}(\mu^{(j)})$ and $\mu^{(j)}$. Towards this goal, the weight corresponding to $\tilde{v}(\mu^{(j)})$ has to be much larger relative to the other weights.

Remark 3. Assuming that the neural networks trained well, due to the pointwise orthogonalization, we would expect the matrix $\mathbf{A}(\mu)$ to be close in norm to a diagonal matrix. Moreover, if they are exactly orthogonal pointwise (which is difficult to observe in practice), then $\mathbf{A}(\mu)$ is diagonal with

$$A(\mu)_{kk} = \sum_{i=1}^m w_e(\mu_i; \mu) \|M_{\mu^{(k)}}(\tilde{v}_i)\|_2^2,$$

so that, solving the system $\mathbf{A}(\mu)\theta = \mathbf{b}(\mu)$ exactly yields

$$\theta_k^{(j)}(\mu) = \frac{\sum_{i=1}^m w_e(\mu_i; \mu) \langle M_{\mu^{(k)}}(\tilde{v}_i), U^\top \mathbf{N}(v_i; \mu) \rangle}{\sum_{i=1}^m w_e(\mu_i; \mu) \|M_{\mu^{(k)}}(\tilde{v}_i)\|_2^2}.$$

Consequently, this gives

$$\hat{\mathbf{N}}^{(j)}(\tilde{v}; \mu) = \sum_{k=1}^j \frac{\sum_{i=1}^m w_e(\mu_i; \mu) \langle M_{\mu^{(k)}}(\tilde{v}_i), U^\top \mathbf{N}(v_i; \mu) \rangle}{\sum_{i=1}^m w_e(\mu_i; \mu) \|M_{\mu^{(k)}}(\tilde{v}_i)\|_2^2} M_{\mu^{(k)}}(\tilde{v}).$$

Interpreting the sums in the coefficient $\theta_k^{(j)}$ as quadrature rules for an L^2 inner product, we notice that $\hat{\mathbf{N}}^{(j)}$ is the sum of approximate projections of $U^\top \mathbf{N}(v; \mu)$ onto the orthogonal neural networks $M_{\mu^{(k)}}$, $k = 1, \dots, j$.

3.3. Choice of the Weights and Stopping Criteria. There are many different ways of defining stopping criteria or the most appropriate weighting rules. One could exploit the knowledge of the system and its behavior, or the rule of thumb given in the first remark, but in general this is problem specific (e.g., dependent on the number of snapshots available, the complexity of the solution as parameters vary, etc.). Let us now describe possible ways to choose the weights for the least square procedures. The error weights $w_e(\mu_i; \mu)$ we choose to find the $\theta(\mu)$ coefficients can be interpreted as quadrature weights. We can also interpret training weights $w_t^{(j)}(\mu_i)$ as quadrature weights, but because we want the neural network architectures to be small so that their evaluation is fast in the online phase, it is not necessarily best to use “optimal” quadrature weights. For example, if the parameters μ_i are equi-spaced, then when the neural network is too small and

all training points are weighted equally, the network has difficulty fitting any of the data at all. Instead, it may be preferable to give nonzero weights to only a small number of parameters μ_i . For example, one could set $w_t^{(j)}(\mu_i) := \delta_{\mu_i \mu^{(j)}}$ to localize the network training. Another option would be a smooth approximation of the Kronecker delta, for some scalar parameters $C, \zeta > 0$

$$w_t^{(j)}(\mu_i) := C \exp \left(-\zeta \|\mu_i - \mu^{(j)}\|_2^2 \right).$$

In the numerical experiments, we specify the weights in each example. Let us now focus on the different choices to implement stopping criteria for the NEIM algorithm. Within our iterative methods, as we will show in the next section, a measure of the error is provably non-increasing for each subsequent iteration, thus, we can stop NEIM whenever this error is sufficiently small. However, we only know that this error is non-increasing, so it is possible that a desired error tolerance is never reached, so this should be combined with a maximum number of iterations. One could also choose to stop NEIM when this error stops decreasing significantly after each iteration, as in the Elbow Method commonly used in clustering algorithms [40].

3.4. Error Analysis. We begin by showing that the measure of error we designed for greedily selecting parameters is non-increasing. We then discuss a local error estimate for NEIM to better identify the different contributions to the error.

Proposition 1 (Error is non-increasing). *The function given by*

$$j \mapsto \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(j-1)}(v_i; \mu)^2, \quad j \in \mathbb{N}$$

is non-increasing for each $\mu \in \mathcal{M}$. In particular, if $w_e(\mu_i; \mu) = \delta_{\mu_i \mu}$, we have that $j \mapsto \mathcal{E}^{(j-1)}(v_i; \mu_i)$, $j \in \mathbb{N}$ is non-increasing.

Proof. At each iteration of NEIM, for every $\mu \in \mathcal{M}$, $\theta_1^{(j)}(\mu), \dots, \theta_j^{(j)}(\mu)$ minimize

$$\sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(j)}(v_i; \mu)^2 = \sum_{i=1}^m w_e(\mu_i; \mu) \left\| U^\top N(v_i; \mu) - \sum_{\ell=1}^j \theta_\ell^{(j)}(\mu) M_{\mu^{(\ell)}}(\tilde{v}_i) \right\|_2^2.$$

Thus, we have that

$$\begin{aligned} \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(j)}(v_i; \mu)^2 &= \arg \min_{\theta \in \mathbb{R}^j} \sum_{i=1}^m w_e(\mu_i; \mu) \left\| U^\top N(v_i; \mu) - \sum_{\ell=1}^j \theta_\ell M_{\mu^{(\ell)}}(\tilde{v}_i) \right\|_2^2 \\ &\leq \sum_{i=1}^m w_e(\mu_i; \mu) \left\| U^\top N(v_i; \mu) - \sum_{\ell=1}^{j-1} \theta_\ell^{(j-1)}(\mu) M_{\mu^{(\ell)}}(\tilde{v}_i) - 0 \cdot M_{\mu^{(j)}}(\tilde{v}_i) \right\|_2^2 \\ &= \sum_{i=1}^m w_e(\mu_i; \mu) \mathcal{E}^{(j-1)}(v_i; \mu)^2, \end{aligned}$$

where in the inequality we choose $\theta = (\theta_1^{(j-1)}, \dots, \theta_{j-1}^{(j-1)}, 0) \in \mathbb{R}^j$. □

Remark 4. Examining the proof of the previous proposition carefully, we see that it can be generalized for more general error measures of the form

$$h_\mu^{(j)}(\theta_1^{(j)}, \dots, \theta_j^{(j)}) := g \left(\sum_{\ell=1}^j \theta_\ell^{(j)} M_{\mu^{(\ell)}}(\tilde{v}_1), \dots, \sum_{\ell=1}^j \theta_\ell^{(j)} M_{\mu^{(\ell)}}(\tilde{v}_m), \mu \right),$$

where $h_\mu^{(j)} : \mathbb{R}^j \rightarrow \mathbb{R}$ has a minimum. Then, the expression in Equation (7) becomes

$$\mu^{(j)} = \arg \max_{\mu \in \mathcal{M} \setminus \{\mu^{(1)}, \dots, \mu^{(j-1)}\}} h_\mu^{(j-1)}(\theta_1^{(j-1)}(\mu), \dots, \theta_{j-1}^{(j-1)}(\mu)),$$

and the expression in equation (9) is replaced by

$$\sum_{\mu \in \mathcal{M}} h_\mu^{(j)}(\theta_1^{(j)}(\mu), \dots, \theta_j^{(j)}(\mu)),$$

where the $\theta_\ell^{(j)}(\mu)$ are to-be-determined by the subsequent steps of NEIM.

Remark 5. The previous proposition still holds if we hold previously computed θ 's constant in the NEIM algorithm. Doing so would not require solving a linear system at each iteration of the NEIM algorithm, but since this linear system is small, we can cheaply recompute it during each iteration.

3.4.1. Local Error Estimate. We now seek a theoretical estimate to explain the sources of error in NEIM. For simplicity, we will consider the case in which the NEIM networks approximate constant vectors, that is, $w_t^{(j)}(\mu_i) > 0$ if and only if $\mu_i = \mu^{(j)}$, and that $w_e(\mu_i; \mu_j) = \delta_{ij}$. We define the matrix of exact vectors as $M^{(k)} = [z^{(1)}(\mu^{(1)}) \dots z^{(k)}(\mu^{(k)})]$, and the matrix of its approximation by neural networks as $\widehat{M}^{(k)}(\mu) = [M_{\mu^{(1)}}(\tilde{v}(\mu)) \dots M_{\mu^{(k)}}(\tilde{v}(\mu))]$. Now define the vector of interpolated θ coefficients resulting from NEIM to be

$$\widehat{\theta}^{(k)}(\mu) = (\widehat{\theta}_1^{(k)}(\mu), \dots, \widehat{\theta}_k^{(k)}(\mu)),$$

and the exact θ coefficients to be

$$\theta^{(k)}(\mu) = \arg \min_{\theta \in \mathbb{R}^k} \|N(\mu) - M^{(k)}\theta\|_2, \quad \mu \in \mathcal{P}.$$

With a little abuse of notation, we denote $N(\mu) := U^\top N(v(\mu); \mu)$, and we want to estimate

$$\mathcal{E}_{\text{NEIM}}^{(k)} := \|N - \widehat{M}^{(k)}\widehat{\theta}^{(k)}\|_{L^2(\mathcal{P})}.$$

Repeatedly applying the triangle inequality and Cauchy-Schwarz inequality, we have

$$\begin{aligned} \mathcal{E}_{\text{NEIM}}^{(k)} &\leq \|N - M^{(k)}\theta^{(k)}\|_{L^2} + \|M^{(k)}\theta^{(k)} - \widehat{M}^{(k)}\theta^{(k)}\|_{L^2} + \|\widehat{M}^{(k)}\theta^{(k)} - \widehat{M}^{(k)}\widehat{\theta}^{(k)}\|_{L^2} \\ &\leq \|N - M^{(k)}\theta^{(k)}\|_{L^2} + \sum_{i=1}^k \|z^{(i)}(\mu^{(i)}) - M_{\mu^{(i)}}(\tilde{v}(\cdot))\|_{L^2} \|\theta_i^{(k)}\|_{L^\infty} \\ &\quad + \sum_{i=1}^k \|M_{\mu^{(i)}}(\tilde{v}(\cdot))\|_{L^2} \|\theta_i^{(k)} - \widehat{\theta}_i^{(k)}\|_{L^\infty} \\ &\leq \underbrace{\|N - M^{(k)}\theta^{(k)}\|_{L^2}}_{\text{Projection error}} + \underbrace{\sum_{i=1}^k \|z^{(i)}(\mu^{(i)}) - M_{\mu^{(i)}}(\tilde{v}(\cdot))\|_{L^2} \|\theta_i^{(k)}\|_{L^\infty}}_{\text{Training error}} \\ &\quad + |\mathcal{P}|^{1/2} \underbrace{\sum_{i=1}^k \|\theta_i^{(k)} - \widehat{\theta}_i^{(k)}\|_{L^\infty}}_{\text{Interpolation error}} + \underbrace{\sum_{i=1}^k \|M_{\mu^{(i)}}(\tilde{v}(\cdot)) - z^{(i)}(\mu^{(i)})\|_{L^2} \|\theta_i^{(k)} - \widehat{\theta}_i^{(k)}\|_{L^\infty}}_{\text{Training/interpolation error interaction}}, \end{aligned}$$

where we used the triangle inequality and the fact that $\|z^{(i)}(\mu^{(i)})\|_2 = 1$.

Note that the projection error above is decreasing to 0 in k if the nonlinearities used for training NEIM are linearly independent. The training error depends on the neural network architectures used, the density of $\mathcal{M}$ in $\mathcal{P}$, and the network training procedure. Finally, the interpolation error also depends on the density of $\mathcal{M}$ in $\mathcal{P}$.

3.5. Comparison with State-of-the-Art Techniques. Here, we qualitatively compare NEIM and the discrete empirical interpolation method (DEIM) [9] to highlight a few advantages of NEIM. We then briefly relate NEIM to DeepONets [28].

In NEIM, we build an approximation of a nonlinearity by greedily choosing parameters corresponding to large error. This is in contrast to DEIM in which one computes a selection operator for the rows of the nonlinearity, and DEIM interpolates exactly at these selected rows. Thus, NEIM allows adaptivity in parameter space, unlike DEIM which is more aptly described as spatial interpolation. NEIM's adaptivity in parameter space would be useful if a given region of parameter space requires more accurate approximation than another region of parameter space, for example.

Now for both NEIM and DEIM, if the nonlinearity acts componentwise on its input, then the complexity of evaluating the nonlinearity depends only on the reduced dimension r . However, DEIM is not as efficient if the nonlinearity does not act componentwise on its input. NEIM, on the other hand, allows efficient computation, even for nonlinearities which do not act componentwise. This is due to the fact that DEIM performs exact evaluation of the nonlinearity at selected rows, unlike NEIM which does not rely on the explicit form of the

nonlinearity, instead approximating simultaneously both the nonlinearity and a nonlinear projection of the nonlinearity.

Because NEIM does not rely on the explicit form of the nonlinearity and instead only data, it is also possible to use NEIM in a strictly data-driven context without explicitly evaluating the nonlinearity. This is not possible with DEIM.

It should be noted, however, that in cases where both NEIM and DEIM can be used to efficiently reduce the time complexity of evaluating nonlinearities, DEIM will typically exhibit faster decay of error, though there are subtleties in this regard (see the numerical results for further discussion of this point). This comparison is summarized in Table 1.

TABLE 1. Properties of the Neural Empirical Interpolation Method (NEIM) and the Discrete Empirical Interpolation Method (DEIM).

NEIM	DEIM
Greedy in parameter space	Greedy in spatial domain
No reliance on explicit form of nonlinearity	Exact evaluation of nonlinearity at selected rows
Nonlinear projection	Linear projection
Efficient for nonlocal and local nonlinearities	Efficient for only local nonlinearities

We now briefly relate NEIM to DeepONets. DeepONets are a specific neural network architecture designed to approximate nonlinear operators [28]. NEIM is related to DeepONets in that NEIM is essentially just a vector-valued DeepONet structured and trained in a greedy fashion when the θ coefficients are predicted by a neural network. Indeed, the θ coefficients in NEIM correspond to the trunk net of a DeepONet, while the vector-valued neural networks in the NEIM expansion correspond to the branch nets of a stacked DeepONet. A sketch of the architecture is given in Figure 1. This suggests that analysis of DeepONets may be modified to analyze NEIM as well.

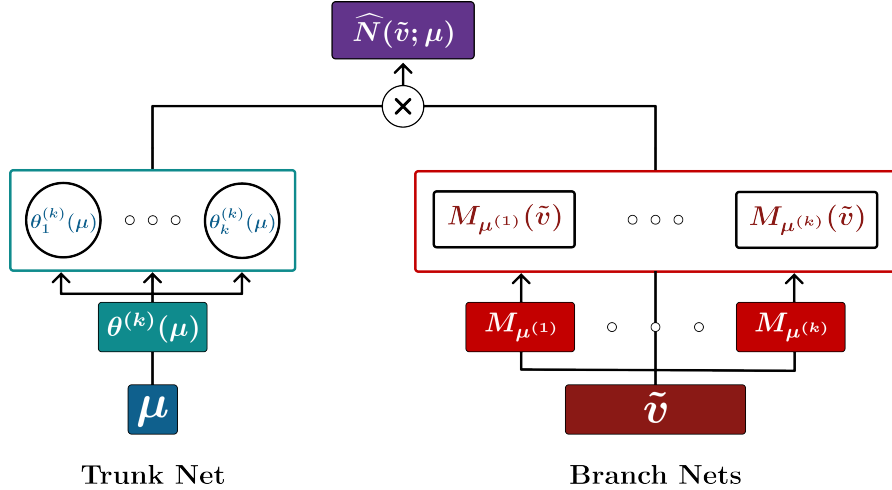


FIGURE 1. DeepONet schematic of NEIM.

4. NUMERICAL EXAMPLES

We now present some numerical examples to demonstrate the performance of NEIM¹. We use the open source library MLniCS [29], based on RBniCS [39] and FEniCS [27], for our physics-informed neural network example. For each experiment, we use PyTorch's learning rate scheduler and the Adam optimizer for training the NEIM networks. The other parameters for the training and NEIM network architecture are listed in Table 2.

¹Code for all examples is available at <https://github.com/maxhirsch/NEIM>.

Example	Hidden Layers	Neurons in Hidden Layer	Epochs
Solution Independent	1	1	20000
Solution Dependent	1	50	10000
PINN	1	10	10000
Liquid Crystal Q, r	1	30	10000
Liquid Crystal θ	2	30	30000

TABLE 2. Architectures exploited for the numerical examples showing the number of hidden layers for each network in the NEIM expansion, the number of neurons in each hidden layer, and the number of epochs for each network in the expansion.

For the liquid crystal experiment, our two NEIM approximations used the same training parameters, which we list in the “Liquid Crystal Q, r ” row. The “Liquid Crystal θ ” row in the table lists the parameters for training the neural network which outputs the prediction of $\theta(\mu)$. We tested other interpolation methods for $\theta(\mu)$ in the other experiments depending on the complexity of the benchmark. While radial basis function interpolation could be used, it did not offer much improvement in our tests, so we used piecewise cubic and piecewise linear interpolation for the prediction of $\theta(\mu)$ instead. In what follows, the “number of modes” used in DEIM will refer to the number of components of the nonlinearity which DEIM selects to interpolate. In NEIM, the “number of modes” will simply refer to the number of terms in the NEIM expansion.

4.1. Solution Independent Function Approximation. In this first example, we approximate a parameterized function that can be thought of as the forcing term in a parameterized PDE discretization. Let us consider $s : \Omega \times \mathcal{P} \rightarrow \mathbb{R}$ defined by $s(x; \mu) = (1 - x) \cos(3\pi\mu(x + 1))e^{-(1+x)\mu}$, where $\Omega = [-1, 1]$ and $\mu \in \mathcal{P} = [1, \pi]$, and the Poisson problem defined by

$$(11) \quad \begin{cases} \Delta v(x) = s(x; \mu), & x \in \Omega, \\ v(-1) = v(1) = 0. \end{cases}$$

Now let $\mathbf{x} = [x_1, \dots, x_n]^\top \in \mathbb{R}^n$ with x_i equally spaced points in Ω for $i = 1, \dots, n$, with $n = 100$, and define the forcing $f : \mathcal{P} \rightarrow \mathbb{R}^n$ by $f(\mu) = [s(x_1; \mu), \dots, s(x_n; \mu)]^\top \in \mathbb{R}^n$ for $\mu \in \mathcal{P}$. The goal is to solve the equation

$$h^{-2} A v_{2:n-1} = f(\mu)_{2:n-1},$$

coming from a finite difference approximation of (11), where $v_1 = v_n = 0$, $h^{-2} = 30$, and

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{(n-2) \times (n-2)}.$$

Here, the subscript in $v_{2:n-1}$ and $f(\mu)_{2:n-1}$ denote the vectors in $\mathbb{R}^{n-2}$ containing the components 2, 3, $\dots$, $n-1$ of v of $f(\mu)$, respectively.

Solving this equation we obtain the snapshots $v(\mu_j)$ for $\{\mu_j\}_{j=1}^m \in \mathcal{P}$, with $m = 51$ snapshots, and perform a POD to obtain the matrix U_r of the first r left singular vectors. The parameters $\{\mu_j\}_{j=1}^m$ are selected as equally spaced points in $\mathcal{P} = [1, \pi]$. Here, we aim at comparing the performance of NEIM and DEIM approaches to approximate the nonlinearity $U_r^\top N(U_r \tilde{v}; \mu) \equiv U_r^\top f(\mu)$.

By looking at the POD singular values to obtain the matrix U_r in Figure 2, we observe that for $r \geq 30$ they are numerically 0. Since these are not meaningful for the expansion, we choose $r = 30$. We now compute a NEIM approximation, where we take all weights equal to 1, and we compare it with a DEIM approach. Because the NEIM neural networks in this case approximate a constant vector-valued function at each iteration as $f(\mu)$ is independent of v , we also include a comparison to “exact” NEIM, that is, the result of NEIM when the neural networks are replaced with the exact constant vectors $M_{\mu^{(i)}}(\tilde{v}) \equiv c_i$ for some $c_i \in \mathbb{R}^r$. This way, we can analyze the performance of NEIM in relation to the accuracy of the different components in the expansion.

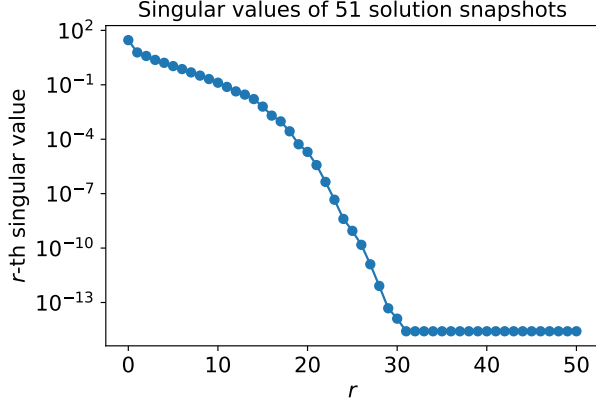


FIGURE 2. Singular values of snapshot matrix in experiment 4.1.

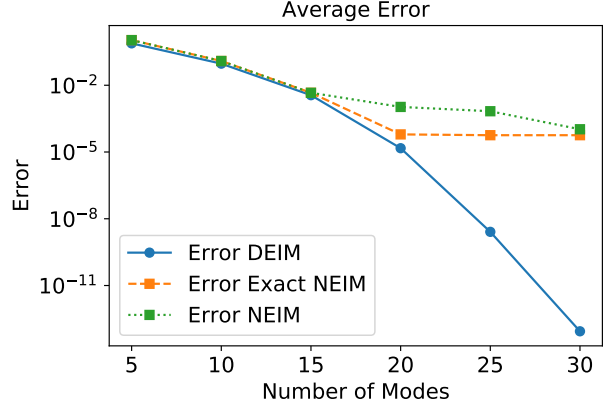


FIGURE 3. NEIM and DEIM average absolute error by number of modes in 4.1.

In Figure 4, we see how the error in NEIM and DEIM develop as the number of modes used increases. The number of modes in DEIM is the number of singular vectors retained in the DEIM POD of the nonlinearity, while the number of modes in NEIM is the number of neural networks in the NEIM expansion. The error for all methods decreases with respect to the number of modes used. For larger numbers of modes, reaching machine precision for the singular values, DEIM outperforms NEIM, but when considering only fewer modes, NEIM performance are remarkably great over the whole parameter space. Indeed, for $r < 20$ the neural empirical interpolation method shows small error at the parameters that have been selected, outperforming DEIM for many parameter values. We remark that NEIM results in small error at the selected parameters $\mu^{(j)}$ primarily because for those parameters we obtain a best fit to $v \mapsto U^\top N(v; \mu^{(j)})$, while the nonlinearities $v \mapsto U^\top N(v; \mu_i)$ for $\mu_i \neq \mu^{(j)}$ may not be close to the span of the trained networks in the NEIM expansion. DEIM seems to have roughly the same error across the parameter space for each number of modes used, while NEIM shows error of different order of magnitude. As expected, the error for the exact version of NEIM, exploiting the exact constant vectors as modes, is smaller than that of the normal NEIM when using a large number of modes. Despite this, we remark that the accuracy of NEIM and NEIM “exact” are equivalent for $r < 20$, confirming the consistency of the approach. In Figure 3, we see how the nonlinearity absolute error averaged over parameters changes with respect to the number of modes used in the NEIM and DEIM algorithms. This error is computed as

$$\frac{1}{m_{\text{test}}} \sum_{j=1}^{m_{\text{test}}} \|\hat{f}(\mu) - U_r^\top f(\mu)\|_2,$$

where $m_{\text{test}} = 500$ is the number of parameters for which we compute the error, and $\hat{f}$ is the DEIM or NEIM approximation of $U_r^\top f(\mu)$. We see that the error for NEIM and DEIM decay similarly for small numbers of modes, but the error for NEIM eventually show a plateau, while the error for DEIM continues to decrease.

4.2. Solution Dependent Function Approximation. We now approximate the parameterized nonlinear function $s : \Omega \times \mathbb{R} \times \mathcal{P} \rightarrow \mathbb{R}$ defined by $s(x, v; \mu) = (1 - |x|)e^{-(1+x)v\mu}$, where $\Omega = [-1, 1]$, $\mu \in \mathcal{P} = [1, \pi]$, appearing as a nonlinear term in

$$(12) \quad \begin{cases} \Delta v(x) = s(x, v(x); \mu), & x \in \Omega, \\ v(-1) = v(1) = 0. \end{cases}$$

Now let x_i be equally spaced points in Ω for $i = 1, \dots, n$, with $n = 100$, and in an abuse of notation, let $v(\mu) \in \mathbb{R}^n$ be the solution of the discretized system,

$$(13) \quad \begin{bmatrix} 0 & \dots & 0 \\ \vdots & h^{-2}A & \vdots \\ 0 & \dots & 0 \end{bmatrix} v - f(v; \mu) = 0, \quad \text{with} \quad v_1 = v_n = 0,$$

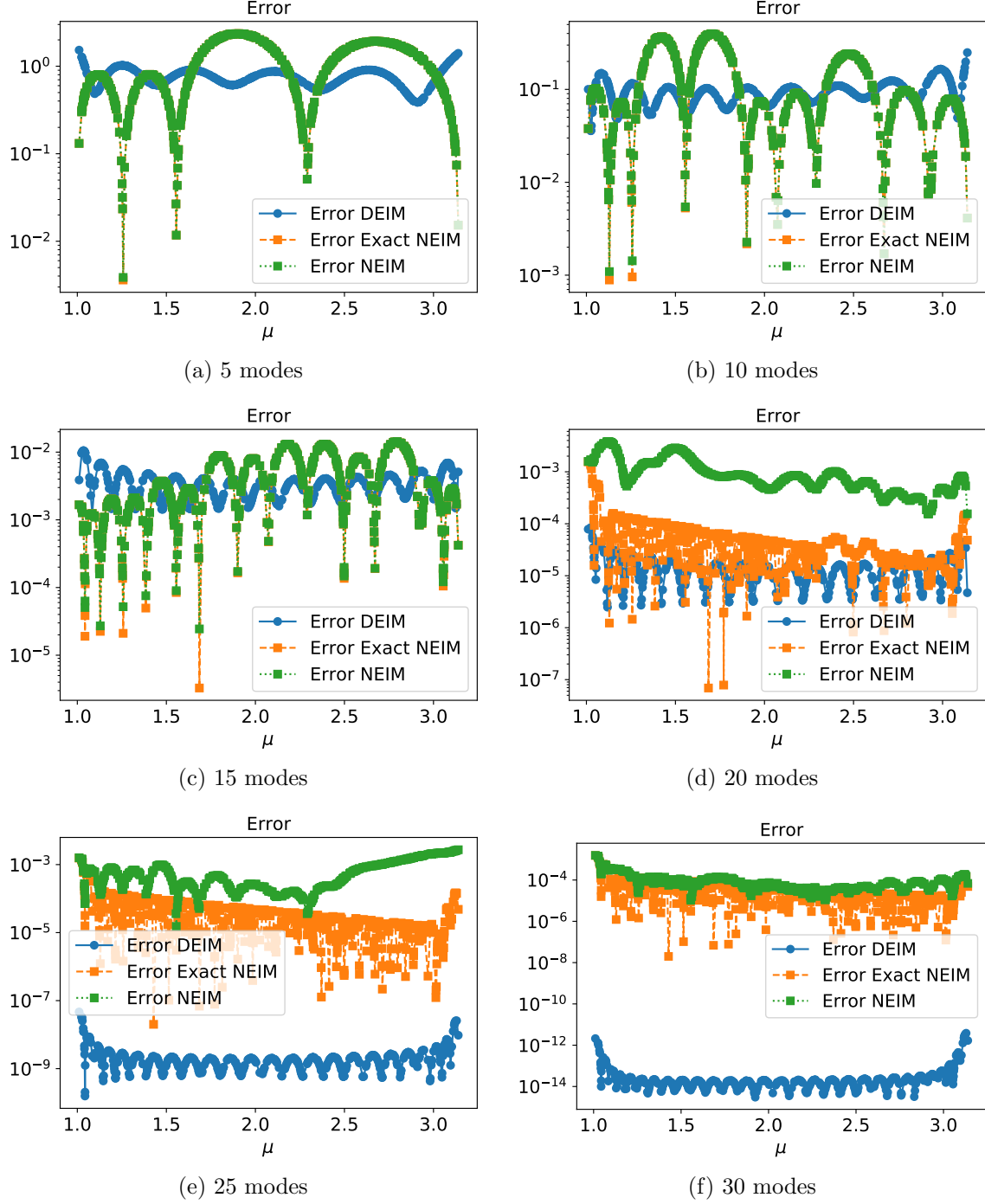


FIGURE 4. NEIM and DEIM absolute error increasing number of modes in 4.1.

where $h^{-2} = 30$ and $f(v; \mu)$ is the nonlinear term in the system defined by $f(v; \mu) = [s(x_1, v_1; \mu) \ \dots \ s(x_n, v_n; \mu)]^\top$. As before, we use $m = 51$ equally spaced parameters in $\mathcal{P} = [1, \pi]$, and perform a POD on the matrix of solution snapshots to obtain a POD basis U_r . The goal now is to approximate $U_r^\top f(U_r \tilde{v}; \mu)$ for $\mu \in \mathcal{P}$. Based on the plot of singular values in Figure 5, we truncate our expansion at $r = 20$ modes. For this test case, we take the weights in the NEIM algorithm to be $w_e(\mu_i; \mu_j) = \delta_{ij}$, $w_t^{(k)}(\mu_i) = 1 \ \forall i, j, k$, and compare the accuracy results of NEIM and DEIM approaches.

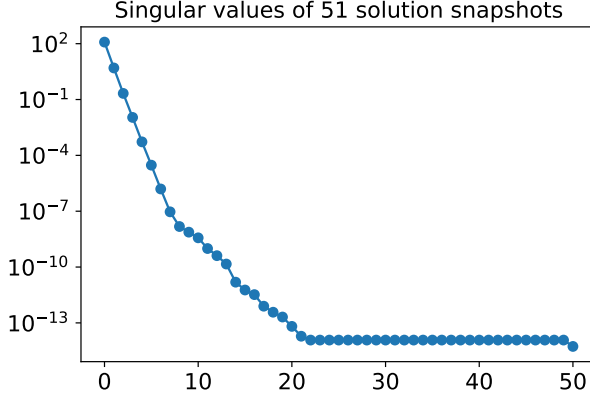


FIGURE 5. Singular values of snapshot matrix for experiment 4.2.

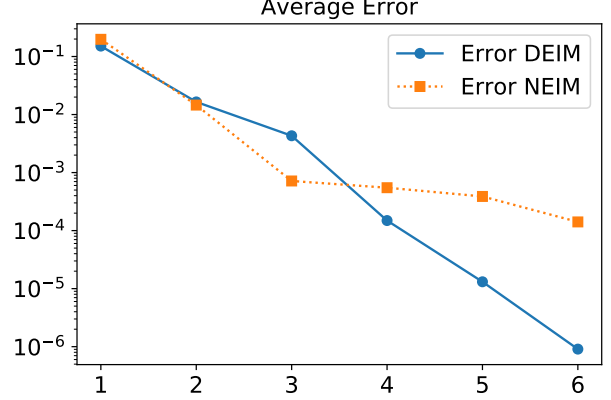


FIGURE 6. NEIM and DEIM average absolute error by number of modes in 4.2.

In Figure 7, we see how the error in NEIM and DEIM develop as the number of modes used increases. We see how the greedy procedure led NEIM to initially select parameters close to the boundary of parameter space, and the errors for NEIM and DEIM both decrease as the number of modes increases, as expected. Also in this case, the error for NEIM is lowest for selected parameters, while both approaches attempt to control the error across the parameter space. While the DEIM error continues to decrease, eventually the NEIM error ceases to decrease significantly, but not before having reached an accuracy of order 10^{-4} . Being a completely data-driven procedure such approximation property is remarkable, especially considering we are not exploiting any physics. As before, we show in Figure 6 how the absolute error averaged over parameters changes with respect to the number of modes used in the NEIM and DEIM algorithms. Here we see more clearly the error decays for both NEIM and DEIM, which exhibit monotone behaviors, even though NEIM error starts flattening out.

Now in Figure 8, we show examples of the learned modes in NEIM and DEIM. The exact nonlinearity in (a) is $U_r U_r^\top f(U_r \tilde{v}(\mu); \mu)$ for $\mu = 1.72$. In (b), we see how the DEIM error is smallest at the selected interpolation points in the DEIM algorithm, as expected. Then plots (c)-(f) are the first six NEIM and DEIM modes for $\mu = 1.72$ and $\mu = 1.00$. In NEIM, these are each of the first six terms in the affine expansion, and in DEIM, these are the residuals approximated by each additional DEIM mode used. The NEIM and DEIM modes in (c) and (d) correspond to $\mu = 1.72$, which was not one of the selected parameters in the NEIM algorithm. We see that both NEIM and DEIM have smooth approximations in the spatial domain, with primarily the first three modes in each being nonzero. Visually, we see that these modes indeed seem to add to the nonlinearity in (a). When $\mu = 1$, which was the second parameter selected by the NEIM algorithm, only the first and second NEIM modes are significantly different from zero, which we expect since NEIM should almost exactly approximate the j th selected parameter with its first j modes. It was also observed that the only significantly nonzero term in the NEIM approximation corresponding to the first selected parameter in the algorithm was the first mode.

Lastly, we compare our NEIM approximation of the nonlinearity with a simple feedforward neural network approximation of $\mu \mapsto U_r^\top f(U_r \tilde{v}(\mu); \mu)$ in Figure 9. We tested feedforward networks containing more parameters than NEIM and also fewer parameters than NEIM. In both cases, NEIM performed better than or comparably to the feedforward networks tested. Thus, in this example, we see that the greedy training algorithm performs better than training everything at once with a single network.

4.3. Nonlinear Elliptic Physics-Informed Neural Network. We now apply NEIM in the context of physics-informed neural networks (PINNs) for reduced order models. The goal is to combine a physics-based learning within a data-driven reduced context. More specifically, one aims at learning the parameter-to-reduced vector mapping via neural network architectures that minimize an unsupervised loss term based on the reduced residual projected onto the linear POD subspace. The setup for the problem is shown in Figure 10. For further details we refer the reader to [22, 10, 33]. In order to embed the reduced physics in the loss for nonlinear problems, it is important that the approximation of the nonlinearity is compatible

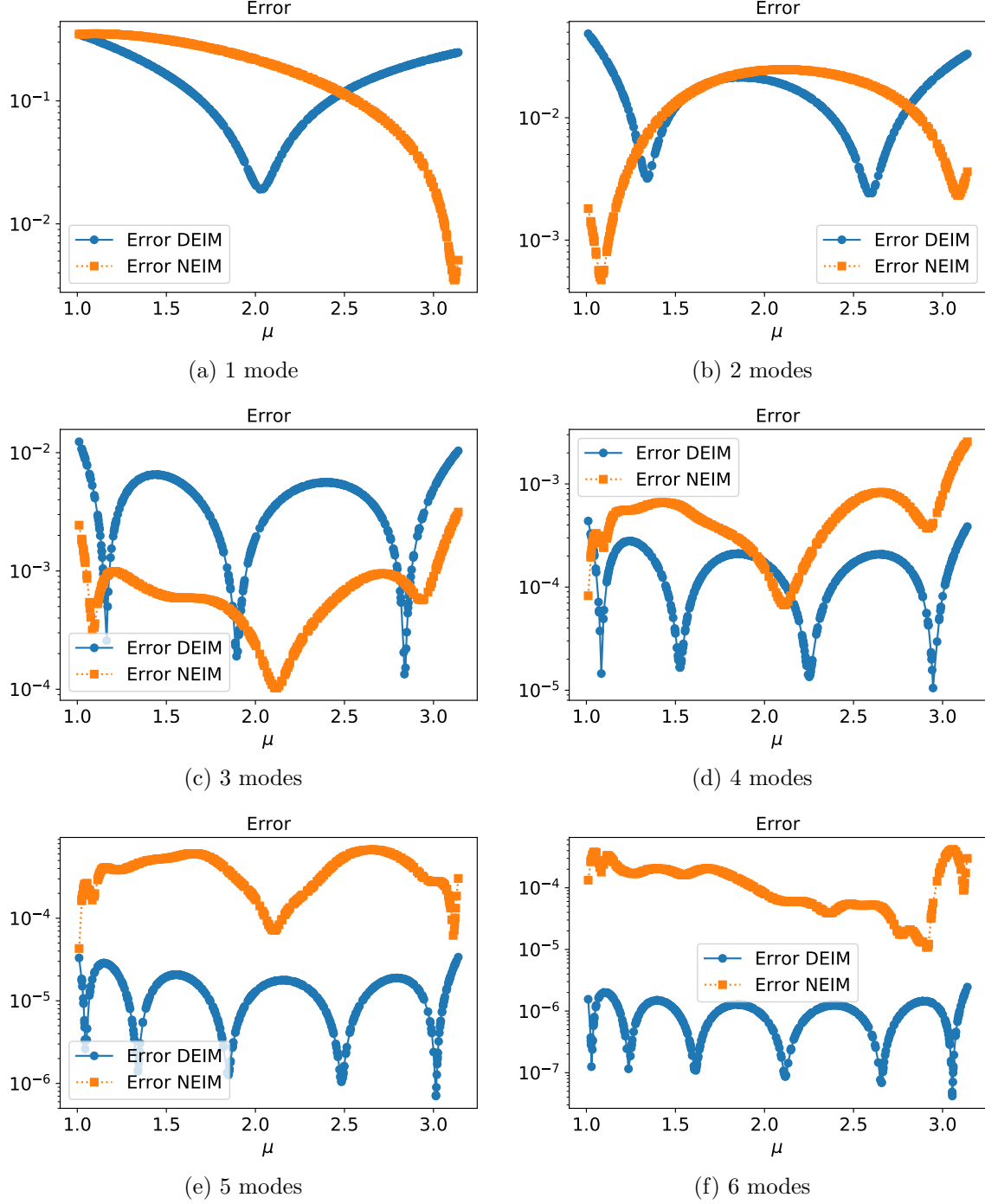


FIGURE 7. NEIM and DEIM absolute errors by parameter for increasing number of modes in experiment 4.2.

with automatic differentiation. When it comes to more complex nonlinearities, exploiting tensor-assembly for polynomial nonlinearities or empirical interpolation approaches can be challenging. In fact, DEIM will require assembling nonlinearities in the online phase of computation, which, if done in standard finite element packages like FEniCS [27], will not be differentiable in deep learning libraries like PyTorch [32]. On the other hand, NEIM is automatically compatible with deep learning libraries because our approach is itself a neural network with automatic differentiation included by definition.

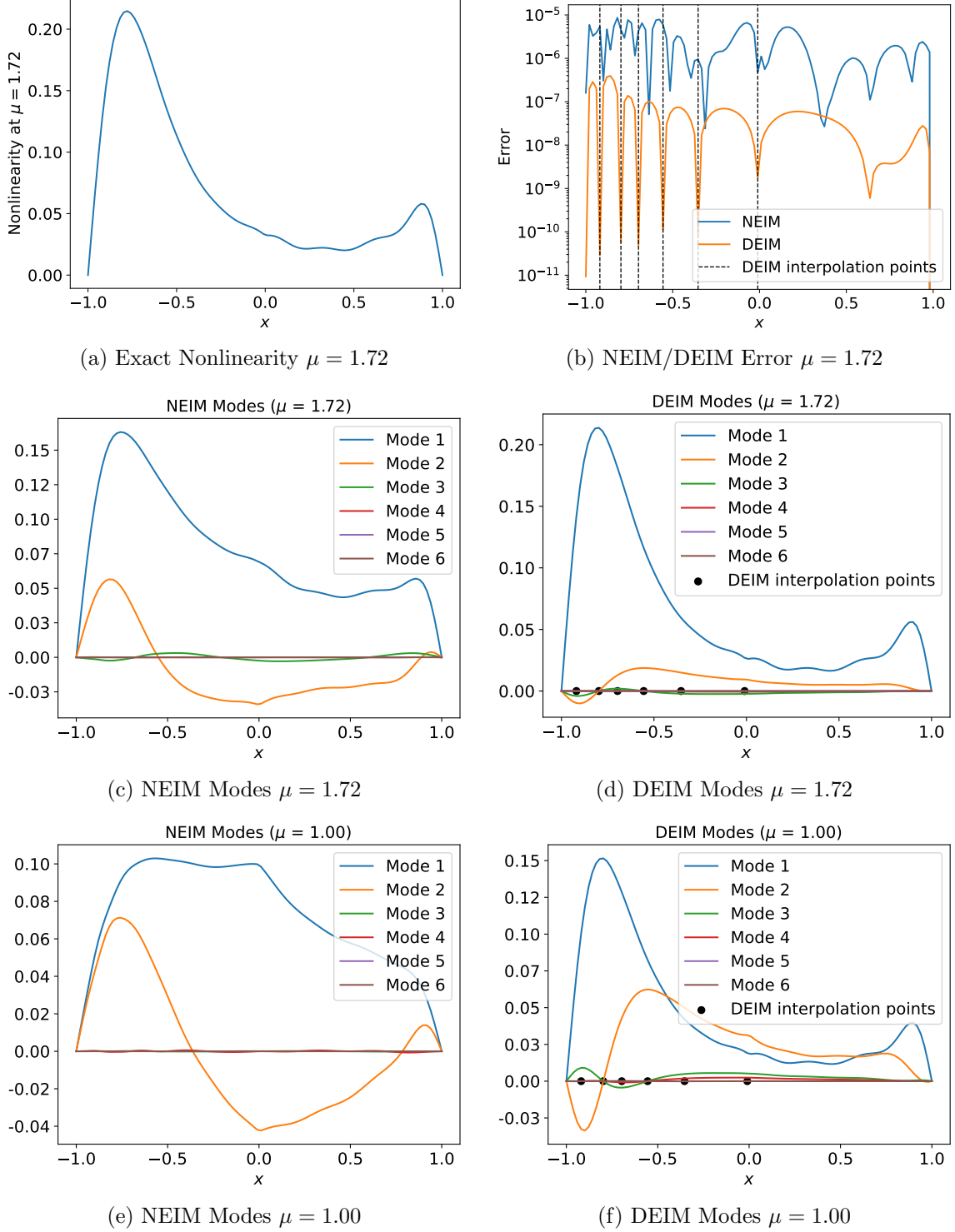


FIGURE 8. NEIM and DEIM modes in experiment 4.2.

Using physics-informed neural networks for model order reduction with NEIM also leads to a completely data-driven and interpretable method for reduced order models. DEIM, on the other hand, still requires the finite element high fidelity operators, and thus prevents its usage in combination with black-box solvers.

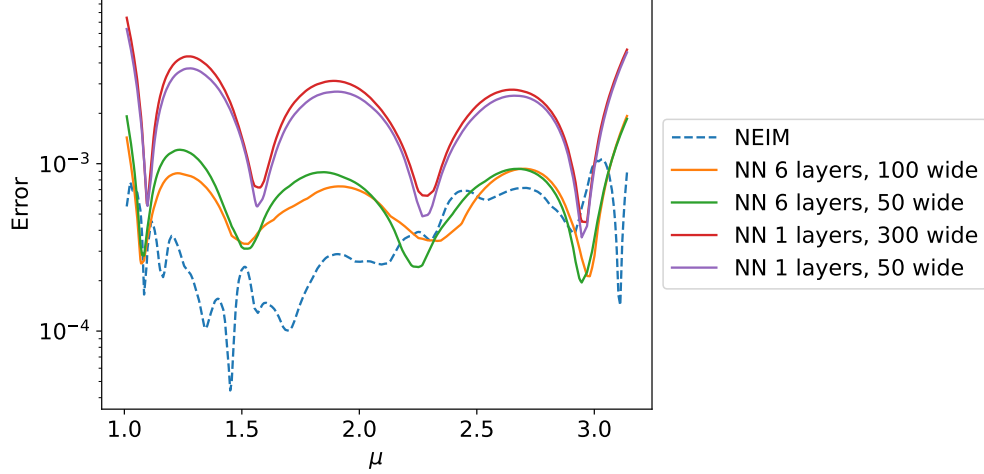


FIGURE 9. Absolute error of NEIM approximation of nonlinearity versus basic neural network approximation for various architectures in experiment 4.2.

Let us consider a nonlinear elliptic problem in a two-dimensional spatial domain $\Omega = (0, 1)^2$. The strong formulation of the problem is given by: for a given parameter $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \mathcal{P} = [0.01, 10.0]^2$, find $v(\boldsymbol{\mu})$ such that

$$-\Delta v(\boldsymbol{\mu}) + \frac{\mu_1}{\mu_2} (\exp^{\mu_2 v(\boldsymbol{\mu})} - 1) = 100 \sin(2\pi x) \sin(2\pi y),$$

with $v(\boldsymbol{\mu})$ satisfying homogeneous Dirichlet boundary conditions.

We exploit the finite element method as our high fidelity solver to discretize the PDE. Then, we perform a POD on $m = 100$ equispaced points in parameter space to obtain U_r . In this case, our goal is to approximate $U_r^\top f(U_r \tilde{v}; \boldsymbol{\mu})$, where f is given by $f(v; \boldsymbol{\mu})_i = \frac{\mu_1}{\mu_2} (\exp(\mu_2 v_i) - 1) \Delta x_i$, $i = 1, \dots, n$, and Δx_i is the volume $\int \varphi_i d\mathbf{x}$ of the piecewise linear finite element basis test function φ_i at node i . In other words, the nonlinearity we aim at approximating is a low order quadrature rule for the nonlinear integral term in the FE formulation of the problem given by $\int \frac{\mu_1}{\mu_2} (\exp(\mu_2 v) - 1) \varphi_i d\mathbf{x} \approx \frac{\mu_1}{\mu_2} (\exp(\mu_2 v_i) - 1) \int \varphi_i d\mathbf{x}$. The weights we use for NEIM are given by $w_e(\mu_i; \mu_j) = \delta_{ij}$ and

$$w_t^{(k)}(\mu_i) = \begin{cases} 1 & \text{if } \|\mu_i - \mu^{(k)}\|_2 \leq 1.75, \\ 0 & \text{otherwise.} \end{cases}$$

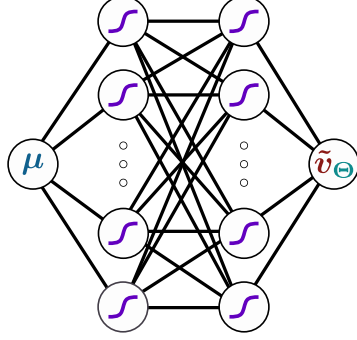
Now we train a physics-informed neural network using DEIM, which we denote by PINN-DEIM, to solve the problem. The physics-informed neural network takes in the parameter for the problem and outputs an estimate of the reduced order solution vector. It is trained by minimizing the sum of the squared norm of the discretized equation residual. We similarly do this for a PINN using NEIM, which we call PINN-NEIM. Each iteration of training is slower for PINN-NEIM than for PINN-DEIM due to the need to backpropagate through the NEIM neural networks, but the loss generally decreases more per iteration for PINN-NEIM. For comparison between the two networks, however, we use 1000 training iterations and two hidden layers with 60 neurons each, for both methods.

We now compare the errors of the two methods on a test set of $m_{\text{test}} = 100$ uniformly random test points. The training and test sets are shown in Figure 11. We note that in a ROM context, one would like to have the discretization of the parameter space as sparse as possible. In general, if we refine our sample for the training set, we expect that the POD would capture more information and that the greedy step in the NEIM algorithm would select more representative points with large error, leading to a more accurate approximation. In this case, we expect that the trend in the error as the number of NEIM modes increases should remain the same.

The relative error to the high fidelity solution averaged over the test set for the PINN-DEIM is about 6.12×10^{-2} , while for the PINN-NEIM, it is about 9.55×10^{-3} . Here we compute the error with the Euclidean

NEIM for PINN

Goal: Approximate parameter-to-POD-coefficients map $\mu \mapsto \tilde{v}_{\Theta}(\mu)$

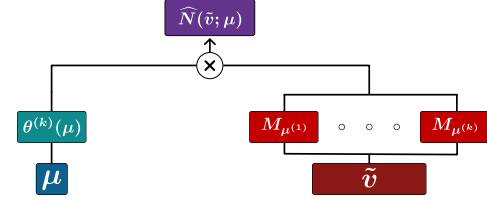


- μ parameters
- Θ neural network weights
- $\mathcal{S}$ activation on hidden layer nodes
- $\tilde{v}_{\Theta}$ solution coefficients

1

Compute NEIM approximation

$$(\tilde{v}, \mu) \mapsto \hat{N}(\tilde{v}; \mu) \approx U_r^\top f(U_r \tilde{v}; \mu)$$



2

Exploit NEIM approximation in PINN loss

Find Θ^* such that $\mathcal{J}(\Theta^*) = \min \mathcal{J}(\Theta)$, where

$$\mathcal{J}(\Theta) = \sum_{\mu \in \Upsilon} \|A \tilde{v}_{\Theta(\mu)} + \hat{N}(\tilde{v}_{\Theta(\mu)}; \mu) - g\|^2$$

with g reduced forcing term, A reduced stiffness matrix, and Υ training set.

3

Obtain the approximation $\mu \mapsto \tilde{v}_{\Theta^*}(\mu)$

FIGURE 10. Setting for NEIM with physics-informed neural networks in experiment 4.3.

relative error in MLniCS [29],

$$\frac{1}{m_{\text{test}}} \sum_{j=1}^{m_{\text{test}}} \frac{\|\hat{v}(\mu_j) - v(\mu_j)\|_2}{\|v(\mu_j)\|_2}.$$

In Figure 12, we plot the PINN-DEIM and PINN-NEIM solutions, and the error fields obtained by the difference between the approximated and the high fidelity solutions for $\mu = (5.5, 5.5)$. As we can see from the plots, the error patterns for both approaches are aligned with the physical phenomena, but the magnitude of the DEIM error is larger. Based on these results, we can see that NEIM, despite being a data-driven approach, is capable of providing meaningful reconstructions of the nonlinear terms, resulting in a competitive approach with respect to DEIM in the PINN context.

4.4. Q-Tensor Model of Liquid Crystals Finite Elements. Finally, we consider a nonlinear parabolic problem in a two-dimensional spatial domain Ω . The problem describes the evolution of liquid crystal

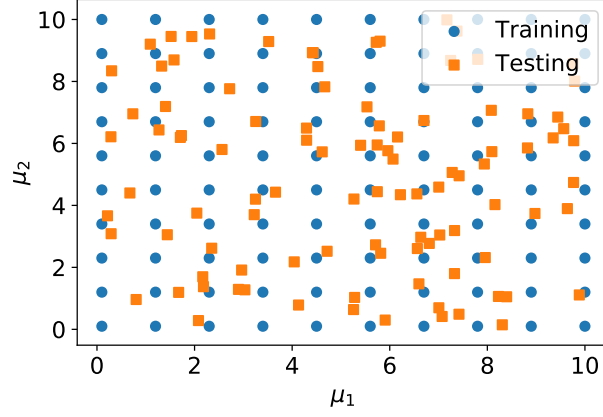
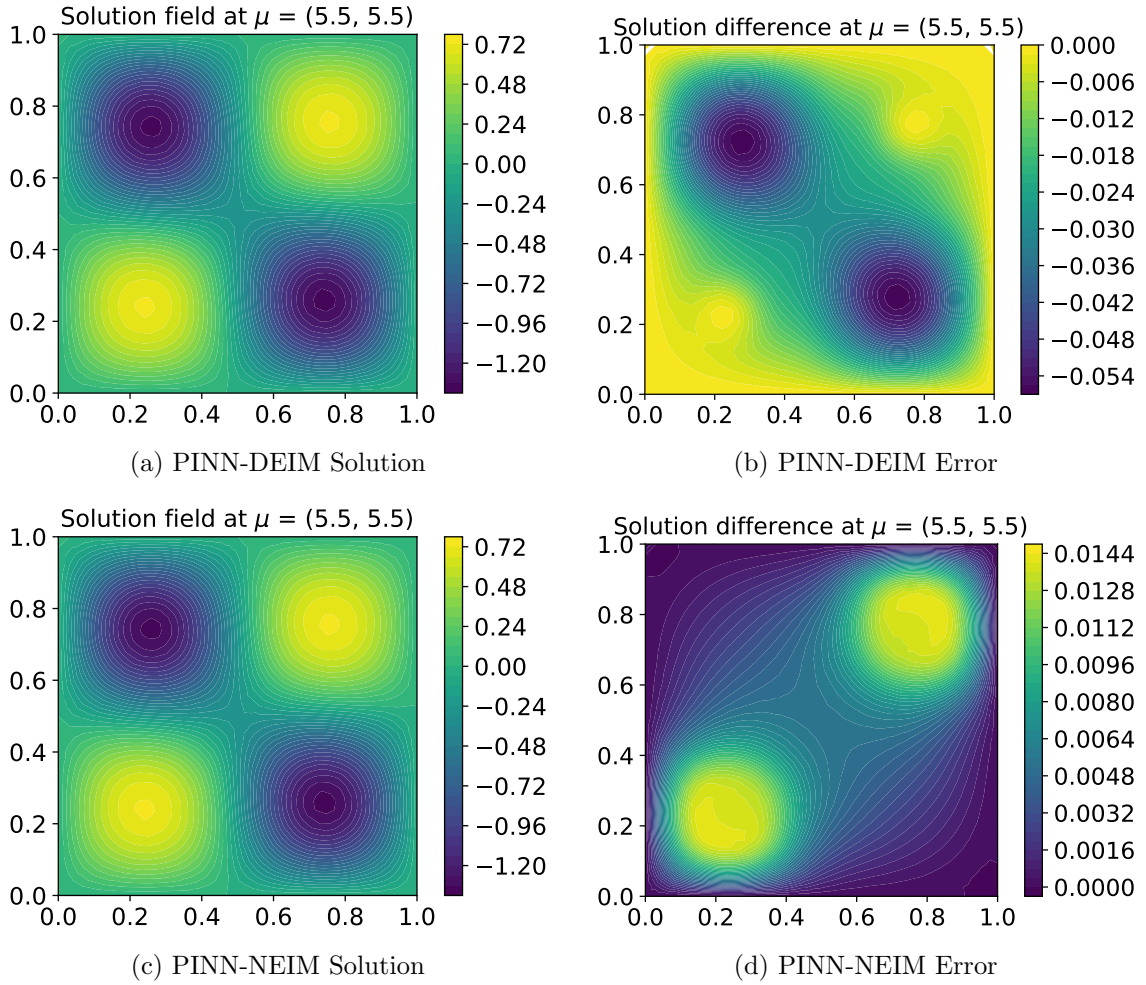


FIGURE 11. Training and testing parameters for experiment 4.3.

FIGURE 12. PINN-DEIM and PINN-NEIM solutions and errors for $\mu = (5.5, 5.5)$ for experiment 4.3.

molecule orientations [19] by means of the coupled matrix-valued PDE:

$$(14) \quad \begin{cases} Q_t &= M (L\Delta Q - rP(Q)) \\ r_t &= \langle P(Q), Q_t \rangle_F, \end{cases}$$

where the unknowns are the 2×2 symmetric and trace-free Q-tensor and the scalar auxiliary variable r , $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product, and

$$(15) \quad P(Q) = \frac{aQ - b[Q^2 - (\text{tr}(Q^2)/2)I] + c \text{tr}(Q^2)Q}{\sqrt{2 \left(\frac{a}{2} \text{tr}(Q^2) - \frac{b}{3} \text{tr}(Q^3) + \frac{c}{4} \text{tr}^2(Q^2) + A_0 \right)}}.$$

Because Q is symmetric and trace-free [19], this problem only depends on three variables: Q_{11} , Q_{12} , and r . To recast the problem in the ROM setting, we consider a parameterized nonlinear term where $a \in [-0.5, 0.5]$ is the varying parameter while holding the other parameters fixed at $M = 1$, $L = 0.1$, $b = 0.5$, $c = 1$, $A_0 = 500$, and we use homogeneous Dirichlet boundary conditions, and initial condition

$$Q_0(x, y) = dd^\top - \frac{\|d\|_2^2}{2} I, \quad d = \frac{[(4-x^2)(4-y^2) \quad \sin(\pi x) \sin(\pi y)]^\top}{\sqrt{1 + [(4-x^2)(4-y^2)]^2 + \sin(\pi x)^2 \sin(\pi y)^2}},$$

where I is the 2×2 identity matrix. We also consider time $t \in [0, 2]$ as a parameter. The parameter vector μ is thus given by $\mu = (t, a)$ on the parameter domain $\mathcal{P} = [0, 2] \times [-0.5, 0.5]$. The authors of [19] show that weak solutions to (14) are weak solutions to the gradient flow

$$Q_t = M \left(L \Delta Q - \frac{\delta \mathcal{F}_B(Q)}{\delta Q} \right),$$

with $\frac{\delta \mathcal{F}_B(Q)}{\delta Q} = aQ - b(Q^2 - \frac{1}{2} \text{tr}(Q^2)I) + c \text{tr}(Q^2)Q$, so we can think of the parameter a as the coefficient of the quadratic term in the quartic double-well bulk potential $\mathcal{F}_B(Q) = \frac{a}{2} \text{tr}(Q^2) - \frac{b}{3} \text{tr}(Q^3) + \frac{c}{4} (\text{tr}(Q^2))^2$. Changing the parameter a affects the depth of the two wells in the double-well potential while keeping the end behavior of the potential the same.

Due to the finite element discretization we exploit, and the complexity of the model, we need to approximate two different nonlinear terms via DEIM/NEIM to recover the solution of the problem. Indeed, the time discretization for the problem is given by

$$\begin{cases} \frac{Q^{n+1} - Q^n}{\Delta t} &= M \left(\frac{L}{2} (\Delta Q^{n+1} + \Delta Q^n) - r^n P(Q^n) \right) \\ r^{n+1} - r^n &= \langle P(Q^n), Q^{n+1} - Q^n \rangle_F, \end{cases}$$

so we see that both $r^n P(Q^n)$ and $\langle P(Q^n), Q^{n+1} - Q^n \rangle_F$ must be approximated. Denote by Q_{11}^n , Q_{12}^n , and r^n the vectors of solutions at time $n\Delta t$, with Δt the time step.

The first term to be approximated is given by

$$f_1(Q_{11}^n, Q_{12}^n, r^n)_i = (r^n)_i P \left(\begin{bmatrix} (Q_{11}^n)_i & (Q_{12}^n)_i \\ (Q_{12}^n)_i & -(Q_{11}^n)_i \end{bmatrix} \right) \Delta x_i, \quad i = 1, \dots, n,$$

where Δx_i is the volume of the finite element basis test function at node i .

The second approximation is for

$$f_2(Q_{11}^n, Q_{12}^n, Q_{11}^{n+1}, Q_{12}^{n+1})_i = \left\langle P \left(\begin{bmatrix} (Q_{11}^n)_i & (Q_{12}^n)_i \\ (Q_{12}^n)_i & -(Q_{11}^n)_i \end{bmatrix} \right), \begin{bmatrix} (Q_{11}^{n+1})_i - (Q_{11}^n)_i & (Q_{12}^{n+1})_i - (Q_{12}^n)_i \\ (Q_{12}^{n+1})_i - (Q_{12}^n)_i & -(Q_{11}^{n+1})_i + (Q_{11}^n)_i \end{bmatrix} \right\rangle_F$$

for $i = 1, \dots, n$. The weights in NEIM are chosen as $w_e(\mu_i; \mu_j) = \delta_{ij}$ and, denoting $\mu_i = (a_i, t_i)$,

$$w_t^{(k)}(a_i, t_i) = \begin{cases} 1 & \text{if } a_i = a_k \text{ and } |t_i - t_k| \leq 5\Delta t, \\ 0 & \text{otherwise,} \end{cases}$$

We fit the reduced order models based on snapshots obtained via a finite element scheme with time step $\Delta t = 0.025$ to final time $T = 2$. The snapshots correspond to 10 equally spaced values of a between -0.5 and 0.5 , and the solutions at each time step for each of these values of a , giving 800 snapshots total. DEIM and NEIM are trained on all of these snapshots but are evaluated with just 4 modes each.

We plot the behavior of the solutions in Figure 13. The blue vectors in the figure denote the orientation of the liquid crystal molecules at a given point, also called the director. In particular, a solution $Q(t, x, y)$ represents a distribution of orientations of liquid crystal molecules at the point (x, y) , so there is an average of that distribution and a measure of how aligned molecules in that distribution are with the average. If $\lambda_1(t, x, y) \geq \lambda_2(t, x, y) = -\lambda_1(t, x, y)$ are the eigenvalues of $Q(t, x, y)$ and $v_1(t, x, y), v_2(t, x, y)$ are the

corresponding eigenvectors, then the blue vectors are the eigenvector $v_1(t, x, y)$. Then the orange contour in the figure denotes how much light passes through the liquid crystal at a given point in space, which is determined by the average orientation of the liquid crystal molecules $v_1(t, x, y)$ and the measure of alignment of molecules in the distribution with the average, $\lambda_1(t, x, y)$. Specifically, the color is given by a color mapping with the scalar quantity $|\lambda_1(t, x, y)(v_1(t, x, y) \cdot (1, 0))|$, so if the molecules are in the vertical direction or are in the isotropic liquid phase (that is, they are not aligned with the average orientation so that $\lambda_1(t, x, y) \approx 0$) then light does not pass through the liquid crystal. We see that qualitatively, the high fidelity, DEIM, and NEIM solutions all have similar contour plots, and the blue director fields are also similar. The contour for DEIM appears more similar to the high fidelity contour than NEIM compared to the high fidelity contour for time $t = 0.375$, but NEIM is more similar at the final time $t = 2$. In both cases, the blue director fields in the DEIM and NEIM plots well-approximate the high fidelity director field, with each plot in the equilibrium state of all horizontal directors.

In Figure 14, we show the relative errors between the components of the DEIM and NEIM models using 4 modes and the high fidelity solution over time for $a = -0.25$, and we can observe that DEIM and NEIM give comparable results. The relative error for r is much smaller than those of Q_{11} and Q_{12} because the scale of r is much larger. Despite the relative error being around 10% error in the worst case, mainly due to the complexity of the model and its time evolution, we already saw in Figure 13 that the solutions were all qualitatively the same, and we are still able to maintain similar accuracy for DEIM and NEIM in this relatively complex application.

In Figure 15, we plot the relative error of Q_{11} , Q_{12} , and r between the solutions computed by DEIM and NEIM and the high fidelity solution for different values of the parameter a . In all plots in the figure, DEIM and NEIM give errors which are comparable in magnitude. The error for NEIM is generally larger for $a > 0$ than for $a < 0$ in the plots of Q_{11} and Q_{12} relative error. This can be explained by the fact that the double-well potential is more complex for $a > 0$. The DEIM relative error, on the other hand, decreases then increases as a increases, with $a = 0$ approximately the minimum error in each case. This is possibly due to the parameters close to the boundary of parameter space being more difficult to approximate. Now, in Figure 16, we plot the absolute and relative errors between the solutions Q_{11} for DEIM and NEIM and the high fidelity solution for parameter pairs (t, a) . Qualitatively, we see that the error for NEIM is larger than that of DEIM, but the two still share a generally similar pattern in the plot of relative error, with positive a and large time being worst-approximated, which reflects the sensitivity to the parameter samples, once again confirming the complexity of the model.

5. CONCLUSIONS

The neural empirical interpolation method is a greedy method of using neural networks to approximate an affine decomposition of a nonlinearity in a reduced order model for a parameterized partial differential equation. It does this by iteratively adding to the NEIM approximation a coefficient times a neural network which fits an orthogonalized residual between the current approximation and the exact reduced nonlinearity. The coefficients are interpolated at the end of this iterative procedure. Thus, the algorithm can be thought of as a greedy method of training a DeepONet.

Because of the greedy nature of the algorithm, we were able to perform a basic error analysis of the method. We showed that the error on the training points used in the algorithm decreases as the number of terms in the approximate affine expansion increases. We also gave a decomposition of the error into a projection error, training error, and interpolation error to understand all the different sources.

We have discussed why NEIM could be preferable to the discrete empirical interpolation method in several contexts. In particular, NEIM is a nonlinear projection, unlike DEIM which is linear. It is also efficient for nonlocal nonlinearities, unlike DEIM which works best for componentwise nonlinearities. Lastly, our method is easily compatible with automatic differentiation, but the discrete empirical interpolation method is not convenient for automatic differentiation when other software like a finite element package is needed to assemble the nonlinearity at each new evaluation.

We have shown via several numerical results that the methodology is effective when dealing with nonlinear terms coming from diverse applications. We showed it in two typical contexts arising from a finite difference discretizations of PDEs and in two more complex examples involving a physics-informed neural network for a nonlinear elliptic problem and a nonlinear parabolic PDE system. In each of these applications, NEIM was comparable to DEIM.

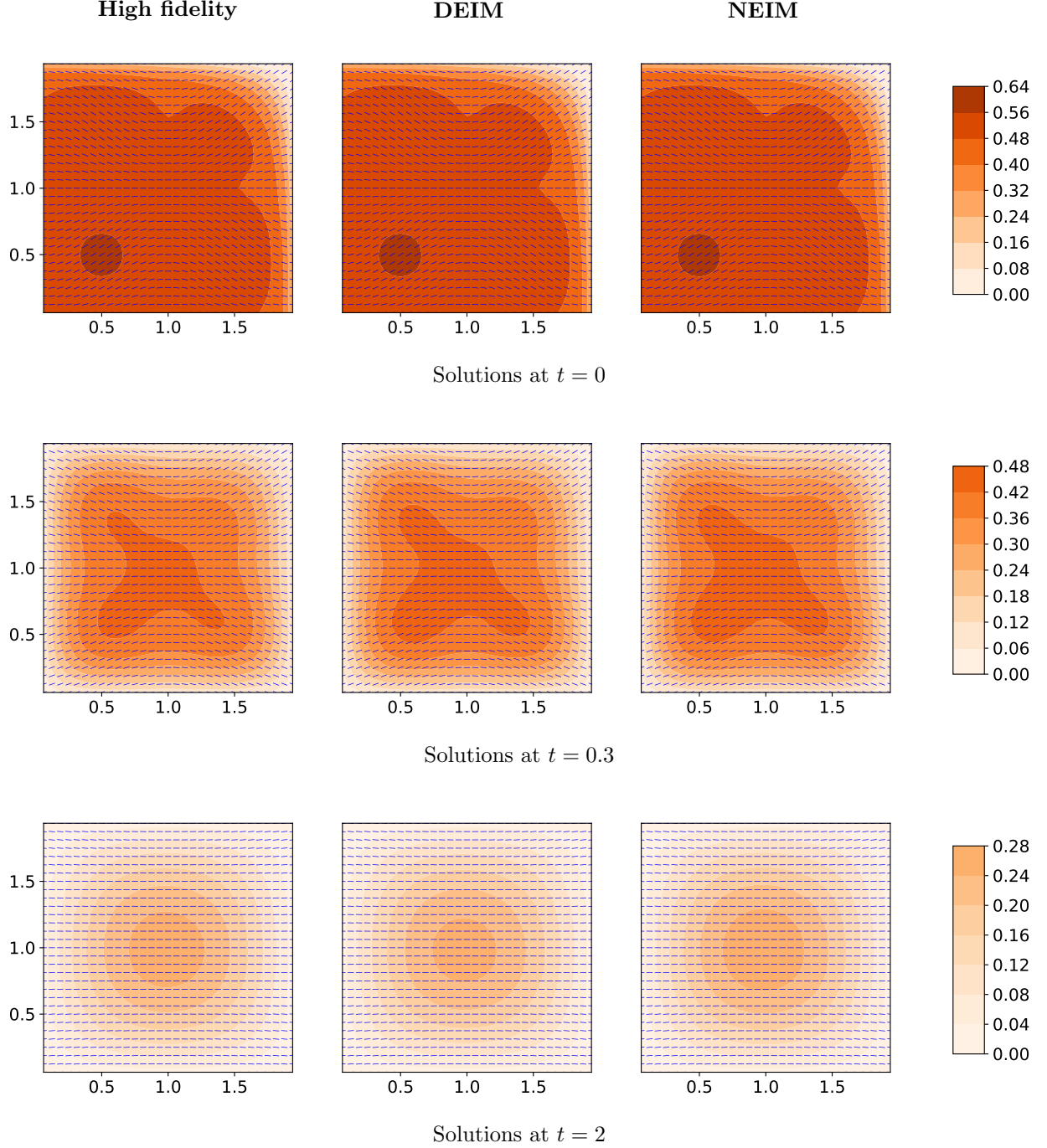
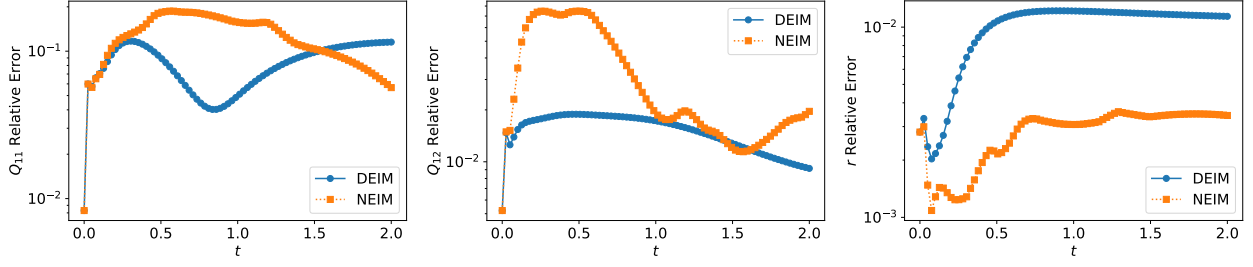
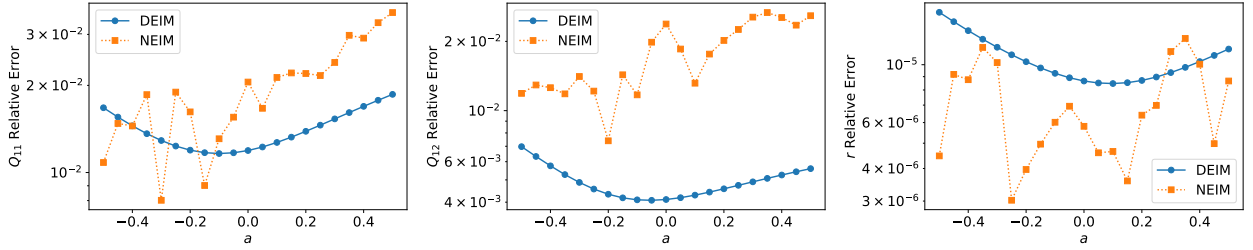
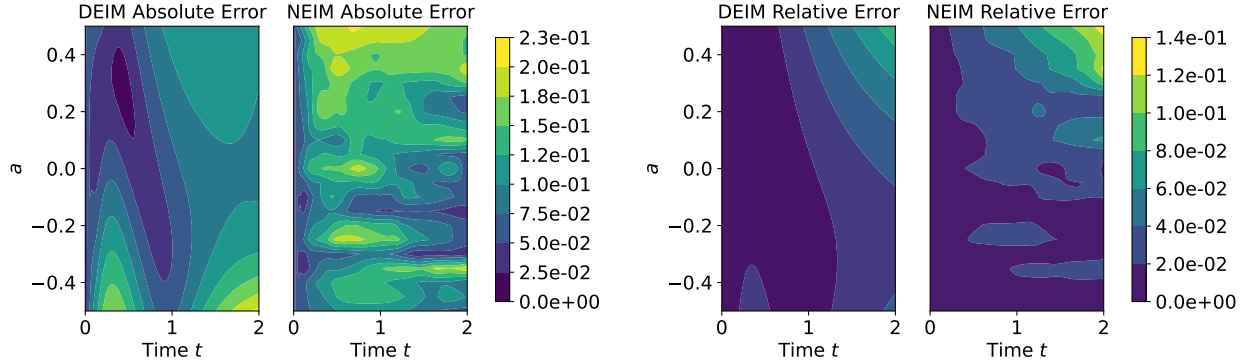


FIGURE 13. High fidelity, DEIM, and NEIM solutions for $a = -0.25$. The blue vectors give the eigenvector corresponding to the positive eigenvalue of the solution Q (the orientation of the liquid crystal). The contour plot represents the intensity of light passing through the liquid crystal.

Lastly, because of its relation to DeepONets, NEIM lends itself to further extension and theoretical analysis. There is much further work to be done on NEIM with respect to optimally training the networks in the NEIM expansion and choosing weights in the NEIM algorithm, the latter being promising given the interpretation of the weights as being quadrature weights. Furthermore, the developed strategy could be used

FIGURE 14. Solution relative errors for $a = -0.25$.FIGURE 15. Relative errors for DEIM and NEIM by a .FIGURE 16. Q_{11} absolute and relative errors for DEIM and NEIM for pairs (t, a) .

to explore further the interconnection between robust numerical algorithms required in the reduced order modelling context, and efficient data-driven methodologies, overcoming the difficulties related to intrusive strategies and extending the range of applicability towards more complex and real-world applications.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 2146752. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. MH also acknowledges the ThinkSwiss Research Scholarship and the EPFL Excellence Research Scholarship. FP acknowledges the “GO for IT” program within the CRUI fund for the project “Reduced order method for nonlinear PDEs enhanced by machine learning”, and the support by Gruppo Nazionale di Calcolo Scientifico (INdAM-GNCS). This work has been conducted within the research activities of the consortium iNEST (Interconnected North-East Innovation Ecosystem), Piano Nazionale di Ripresa e Resilienza (PNRR) - Missione 4 Componente 2, Investimento 1.5 - D.D. 1058 23/06/2022, ECS00000043, supported by the European Union’s NextGenerationEU program.

REFERENCES

- [1] David Amsallem, Matthew J. Zahr, and Charbel Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10):891–916, 2012.
- [2] Harbir Antil, Madhu Gupta, and Randy Price. A Note on Dimensionality Reduction in Deep Neural Networks using Empirical Interpolation Method, 2023.
- [3] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An ‘empirical interpolation’ method: Application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathématique*, 339(9):667–672, 2004.
- [4] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18(1):5595–5637, January 2017.
- [5] Peter Benner, Sefano Grivet Talocia, Alfio Quarteroni, Gianluigi Rozza, Wil Schilders, and Luis Miguel Silveira. *Snapshot-Based Methods and Algorithms*, volume 2. De Gruyter, 2020.
- [6] Peter Benner, Sefano Grivet Talocia, Alfio Quarteroni, Gianluigi Rozza, Wil Schilders, and Luis Miguel Silveira. *System-and Data-Driven Methods and Algorithms*, volume 1. De Gruyter, 2021.
- [7] Nicolas Boullé and Alex Townsend. A Mathematical Guide to Operator Learning, 2023.
- [8] Tan Bui-Thanh, Murali Damodaran, and Karen Willcox. Proper orthogonal decomposition extensions for parametric applications in compressible aerodynamics. In *21st AIAA Applied Aerodynamics Conference*, page 4213, 2003.
- [9] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear Model Reduction via Discrete Empirical Interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.
- [10] Wenqian Chen, Qian Wang, Jan S. Hesthaven, and Chuhua Zhang. Physics-informed machine learning for reduced-order modeling of nonlinear problems. *Journal of Computational Physics*, 446:110666, 2021.
- [11] Ludovica Cicci, Stefania Fresca, and Andrea Manzoni. Deep-HyROMnet: A Deep Learning-Based Operator Approximation for Hyper-Reduction of Nonlinear Parametrized PDEs. *Journal of Scientific Computing*, 93(2):57, 2022.
- [12] Nicola Demo, Dario Coscia, and Gianluigi Rozza. A space-continuous implementation of proper orthogonal decomposition by means of neural networks, 2024.
- [13] Nicola Demo, Marco Tezzele, and Gianluigi Rozza. A non-intrusive approach for the reconstruction of POD modal coefficients through active subspaces. *Comptes Rendus Mécanique*, 347(11), 2019.
- [14] Pedro Díez, Alba Muixí, Sergio Zlotnik, and Alberto García-González. Nonlinear dimensionality reduction for parametric problems: A kernel proper orthogonal decomposition. *International Journal for Numerical Methods in Engineering*, 122(24):7306–7327, 2021.
- [15] Lukas Einkemmer, Jingwei Hu, and Shiheng Zhang. Asymptotic-preserving dynamical low-rank method for the stiff nonlinear boltzmann equation, 2025.
- [16] Nicola Rares Franco, Andrea Manzoni, Paolo Zunino, and Jan S. Hesthaven. Deep orthogonal decomposition: A continuously adaptive data-driven approach to model order reduction, 2024.
- [17] Stefania Fresca, Luca Dede’, and Andrea Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87:1–36, 2021.
- [18] Rudy Geelen and Karen Willcox. Localized non-intrusive reduced-order modelling in the operator inference framework. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2229):20210206, 2022.
- [19] Varun M. Gudibanda, Franziska Weber, and Yukun Yue. Convergence analysis of a fully discrete energy-stable numerical scheme for the Q-tensor flow of liquid crystals. *SIAM Journal on Numerical Analysis*, 60(4):2150–2181, 2022.
- [20] Mengwu Guo and Jan S Hesthaven. Reduced order modeling for nonlinear structural analysis using Gaussian process regression. *Computer methods in applied mechanics and engineering*, 341:807–826, 2018.
- [21] Jan S. Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. SpringerBriefs in Mathematics. Springer International Publishing AG, Cham, 1st ed. 2016 edition, 2015.
- [22] Jan S. Hesthaven and Stefano Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.
- [23] Mariella Kast, Mengwu Guo, and Jan S Hesthaven. A non-intrusive multifidelity method for the reduced order modeling of nonlinear problems. *Computer Methods in Applied Mechanics and Engineering*, 364:112947, 2020.
- [24] Moaad Khamlich, Federico Pichi, and Gianluigi Rozza. Optimal Transport-Inspired Deep Learning Framework for Slow-Decaying Kolmogorov n-Width Problems: Exploiting Sinkhorn Loss and Wasserstein Kernel. *SIAM Journal on Scientific Computing*, pages C235–C264, 2025.
- [25] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.
- [26] Kookjin Lee and Kevin T Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404, 2020.
- [27] Anders Logg, Kent-Andre Mardal, and Garth Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, Heidelberg, 2012.
- [28] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deepnet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar 2021.

- [29] MLniCS. <https://github.com/MLniCS/MLniCS>.
- [30] Oisín M. Morrison, Federico Pichi, and Jan S. Hesthaven. GFN: A graph feedforward network for resolution-invariant reduced operator learning in multifidelity applications. *Computer Methods in Applied Mechanics and Engineering*, 432:117458, 2024.
- [31] Mario Ohlberger and Stephan Rave. Reduced basis methods: Success, limitations and future challenges. *arXiv preprint arXiv:1511.02021*, 2015.
- [32] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.
- [33] Federico Pichi, Francesco Ballarin, Gianluigi Rozza, and Jan S. Hesthaven. An artificial neural network approach to bifurcating phenomena in computational fluid dynamics. *Computers & Fluids*, 254:105813, 2023.
- [34] Federico Pichi, Beatriz Moya, and Jan S. Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs. *Journal of Computational Physics*, 501:112762, 2024.
- [35] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced Basis Methods for Partial Differential Equations: An Introduction*. La Matematica per Il 3+2, 92. Springer International Publishing, Cham, 1st ed. 2016. edition, 2016.
- [36] Alfio Quarteroni and Alberto Valli. *Numerical Approximation of Partial Differential Equations*. Springer-Verlag, 1994.
- [37] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [38] Francesco Romor, Giovanni Stabile, and Gianluigi Rozza. Non-linear manifold reduced-order models with convolutional autoencoders and reduced over-collocation method. *Journal of Scientific Computing*, 94(3):74, 2023.
- [39] Gianluigi Rozza, Francesco Ballarin, Leonardo Scandurra, and Federico Pichi. *Real Time Reduced Order Computational Mechanics: Parametric PDEs Worked Out Problems*, volume 5 of *SISSA Springer Series*. Springer Nature Switzerland, Cham, 2024.
- [40] Erich Schubert. Stop using the elbow criterion for k-means and how to choose the number of clusters instead. *SIGKDD Explor. Newsl.*, 25(1):36–42, jul 2023.
- [41] Tommaso Taddei. A registration method for model order reduction: data compression and geometry reduction. *SIAM Journal on Scientific Computing*, 42(2):A997–A1027, 2020.
- [42] Davide Torlo. Model Reduction for Advection Dominated Hyperbolic Problems in an ALE Framework: Offline, Online Phases and Error Estimator. In Adélia Sequeira, Ana Silvestre, Svilen S. Valtchev, and João Janela, editors, *Numerical Mathematics and Advanced Applications ENUMATH 2023, Volume 2*, pages 409–419, Cham, 2025. Springer Nature Switzerland.