Journal of Computational Physics 363 (2018) 55–78

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Non-intrusive reduced order modeling of nonlinear problems using neural networks

J.S. Hesthaven [a], S. Ubbiali [a,b,*]

[a] École Polytechnique Fédérale de Lausanne (EPFL), Route Cantonale, 1015 Lausanne, Switzerland
[b] Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy

ARTICLE INFO

Article history:
Received 31 October 2017
Received in revised form 29 January 2018
Accepted 19 February 2018
Available online 26 February 2018

Keywords:
Non-intrusive reduced basis method
Proper orthogonal decomposition
Multi-layer perceptron
Levenberg–Marquardt algorithm
Poisson equation
Driven cavity flow

ABSTRACT

We develop a non-intrusive reduced basis (RB) method for parametrized steady-state partial differential equations (PDEs). The method extracts a reduced basis from a collection of high-fidelity solutions via a proper orthogonal decomposition (POD) and employs artificial neural networks (ANNs), particularly multi-layer perceptrons (MLPs), to accurately approximate the coefficients of the reduced model. The search for the optimal number of neurons and the minimum amount of training samples to avoid overfitting is carried out in the offline phase through an automatic routine, relying upon a joint use of the Latin hypercube sampling (LHS) and the Levenberg–Marquardt (LM) training algorithm. This guarantees a complete offline-online decoupling, leading to an efficient RB method – referred to as POD-NN – suitable also for general nonlinear problems with a non-affine parametric dependence. Numerical studies are presented for the nonlinear Poisson equation and for driven cavity viscous flows, modeled through the steady incompressible Navier–Stokes equations. Both physical and geometrical parametrizations are considered. Several results confirm the accuracy of the POD-NN method and show the substantial speed-up enabled at the online stage as compared to a traditional RB strategy.

1. Introduction

Many applications in engineering and the applied sciences involve mathematical models expressed as parametrized partial differential equations (PDEs), in which boundary conditions, material properties, source terms, loads or geometric features of the underlying physical problem are expressed by a parameter $\boldsymbol{\mu}$ [18,23,26]. A list of notable examples includes parameter estimation [6], topology optimization [5], optimal control [31] and uncertainty quantification [29]. In these examples, one is typically interested in a real-time evaluation of an *output of interest* (defined as a functional of the state variable [15]) for many parameter entries, i.e., for many configurations of the problem.

The increasing computational power and the simultaneous algorithmic improvements enable nowadays the *high-fidelity* numerical resolution of complex problems via standard discretization procedures, such as finite difference (FD), finite volume (FV), finite element (FE), or spectral methods [44]. However, these schemes remain prohibitively expensive in many-query and real-time contexts, both in terms of CPU time and memory demand, due to the large amount of degrees of freedom (DOFs) they need to accurately solve the PDE [1]. In light of this, *reduced order modeling* (ROM) methods have received

* Corresponding author.
E-mail addresses: jan.hesthaven@epfl.ch (J.S. Hesthaven), subbiali@phys.ethz.ch (S. Ubbiali).

https://doi.org/10.1016/j.jcp.2018.02.037
0021-9991/

a significant attention in the last decades. The objective of these methods is to replace the full-order system by one of significant smaller dimension, to decrease the computational burden while leading to a controlled loss of accuracy [11].

Reduced basis (RB) methods constitute a well-known and widely-used example of reduced order modeling techniques. They are generally implemented pursuing an offline-online paradigm [33]. Based upon an ensemble of *snapshots* (i.e., high-fidelity solutions to the parametrized differential problem), the goal of the *offline* step is to construct a solution-dependent basis, yielding a reduced space of globally approximating functions to represent the main dynamics of the full-order model [2,11]. For this, two major approaches have been proposed in the literature: proper orthogonal decomposition (POD) [32,49] and greedy algorithms [24]. The former relies on a deterministic or random sample in the parameter space to generate snapshots and then employs a singular value decomposition (SVD) to recover the reduced basis. In the second approach, the basis vectors coincide with the snapshots themselves, carefully selected according to some optimality criterion. As a result, a greedy strategy is typically more effective and efficient than POD, as it enables the exploration of a wider region of the parameter space while entailing the computation of many fewer high-fidelity solutions [23]. However, there exist problems for which a greedy approach is not feasible, simply because a natural criterion for the choice of the snapshots is not available [2].

Once a reduced order framework has been properly set up, an approximation to the *truth* solution for a new parameter value is sought *online* as a linear combination of the RB functions, with the expansion coefficients determined via a projection of the full-order system onto the reduced space [7]. To this end, a Galerkin procedure is the most popular choice.

Despite their established effectiveness, projection-based RB methods do not provide any computational gain with respect to a direct (expensive) approach for complex nonlinear problems with a non-affine dependence on the parameters. This is a result of the cost to compute the projection coefficients, which depends on the dimension of the full-order model. In fact, a full decoupling between the online stage and the high-fidelity scheme is the ultimate secret for the success of any RB procedure [44]. For this purpose, one may recover an affine expansion of the differential operator through the empirical interpolation method (EIM) [3] or its discrete variants [10,40]. However, for general nonlinear problems this is far from trivial.

A valuable alternative to address this concern is represented by *non-intrusive* RB methods, in which the high-fidelity model is used to generate the snapshots, but not in the projection process [11]. The projection coefficients are obtained via interpolation over the parameter domain of a database of reduced order information [9]. However, since reduced bases generally belong to nonlinear, matrix manifolds, standard interpolation techniques may fail, as they cannot enforce the constraints characterizing those manifolds, unless employing a large amount of samples [1,4].

In this work, we develop a non-intrusive RB method employing POD for the generation of the reduced basis and resort to (artificial) neural networks, in particular multi-layer perceptrons, in the interpolation step. Hence, in the following we refer to the proposed RB procedure as the POD-NN method. Being of non-intrusive nature, POD-NN is suitable for a fast and reliable resolution of complex nonlinear PDEs featuring a non-affine parametric dependence. To test this assertion, the POD-NN method is applied to the one- and two-dimensional nonlinear Poisson equation and to the steady incompressible Navier–Stokes equations. Both physical and geometrical parametrizations are considered.

The paper is organized as follows. Section 2 defines the (parametrized) functional and variational framework which is required to develop a finite element solver, briefly outlined in Subsection 2.2. The standard projection-based POD-Galerkin (POD-G) RB method is derived in Section 3. Section 4 discusses components, topology and learning process for artificial neural networks. This is preparatory for the subsequent Section 5, which details the non-intrusive POD-NN RB procedure; both theoretical and practical aspects are addressed. Several numerical results, aiming to show the reliability and efficiency of the proposed RB technique, are offered in Section 6 for the Poisson equation (Subsection 6.1) and the lid-driven cavity problem for the steady Navier–Stokes equations (Subsection 6.2). Finally, Section 7 gathers some relevant conclusions and suggests future developments.

2. Parametrized partial differential equations

Assume $\mathcal{P}_{ph} \subset \mathbb{R}^{P_{ph}}$ and $\mathcal{P}_g \subset \mathbb{R}^{P_g}$ are compact sets, and let $\boldsymbol{\mu}_{ph} \in \mathcal{P}_{ph}$ and $\boldsymbol{\mu}_g \in \mathcal{P}_g$ be respectively the *physical* and *geometrical* parameters characterizing the differential problem, so that $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathcal{P} = \mathcal{P}_{ph} \times \mathcal{P}_g \subset \mathbb{R}^P$, with $P = P_{ph} + P_g$, represents the overall *input vector parameter*. While $\boldsymbol{\mu}_{ph}$ addresses material properties, source terms and boundary conditions, $\boldsymbol{\mu}_g$ defines the shape of the computational domain $\widetilde{\Omega} = \widetilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 1, 2$. We denote by $\widetilde{\Gamma}(\boldsymbol{\mu}_g) = \partial\widetilde{\Omega}(\boldsymbol{\mu}_g)$ the (Lipschitz) boundary of $\widetilde{\Omega}(\boldsymbol{\mu}_g)$, and by $\widetilde{\Gamma}_D(\boldsymbol{\mu}_g)$ and $\widetilde{\Gamma}_N(\boldsymbol{\mu}_g)$ the portions of $\widetilde{\Gamma}(\boldsymbol{\mu}_g)$ where Dirichlet and Neumann boundary conditions are enforced, respectively, with $\widetilde{\Gamma}_D \cup \widetilde{\Gamma}_N = \widetilde{\Gamma}$ and $\mathring{\widetilde{\Gamma}}_D \cap \mathring{\widetilde{\Gamma}}_N = \emptyset$.

Consider a Hilbert space $\widetilde{V} = \widetilde{V}(\boldsymbol{\mu}_g) = \widetilde{V}(\widetilde{\Omega}(\boldsymbol{\mu}_g))$ defined over the domain $\widetilde{\Omega}(\boldsymbol{\mu}_g)$, equipped with the scalar product $(\cdot,\cdot)_{\widetilde{V}}$ and the induced norm $\|\cdot\|_{\widetilde{V}} = \sqrt{(\cdot,\cdot)_{\widetilde{V}}}$. Furthermore, let $\widetilde{V}' = \widetilde{V}'(\boldsymbol{\mu}_g)$ be the dual space of $\widetilde{V}$. Denoting by $\widetilde{G} : \widetilde{V} \times \mathcal{P}_{ph} \to \widetilde{V}'$ the map representing a parametrized nonlinear second-order PDE, the differential (strong) form of the problem of interest reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathcal{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{G}(\widetilde{u}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph}) = 0 \quad \text{in } \widetilde{V}'(\boldsymbol{\mu}_g)\,, \tag{2.1}$$

namely

$$\langle \widetilde{G}(\widetilde{u}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph}),\, v\rangle_{\widetilde{V}',\widetilde{V}} = 0 \quad \forall v \in \widetilde{V}(\boldsymbol{\mu}_g)\,,$$

with $\langle\cdot,\,\cdot\rangle_{\widetilde{V}',\widetilde{V}} : \widetilde{V}' \times \widetilde{V} \to \mathbb{R}$ the duality pairing between $\widetilde{V}'$ and $\widetilde{V}$.

The finite element method requires problem (2.1) to be stated in a weak (or variational) form [43]. To this end, let us introduce the form $\widetilde{g} : \widetilde{V} \times \widetilde{V} \times \mathcal{P} \to \mathbb{R}$, with $\widetilde{g}(\cdot,\,\cdot;\,\boldsymbol{\mu})$ defined as:

$$\widetilde{g}(w,\,v;\,\boldsymbol{\mu}) = \langle \widetilde{G}(w;\,\boldsymbol{\mu}_{ph}),\,v\rangle_{\widetilde{V}',\widetilde{V}} \quad \forall w,\,v \in \widetilde{V}\,.$$

The variational formulation of (2.1) then reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph},\,\boldsymbol{\mu}_g) \in \mathcal{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{g}(\widetilde{u}(\boldsymbol{\mu}),\,v;\,\boldsymbol{\mu}) = 0 \quad \forall v \in \widetilde{V}(\boldsymbol{\mu}_g)\,.$$

2.1. From physical to reference domain

As discussed in the Introduction, a reduced basis method seeks an approximated solution to a problem as a combination of (few) well-chosen basis vectors. These typically result from a suitable combination of a collection of high-fidelity approximations, called *snapshots*. Therefore, when addressing problems defined on variable shape domains, ensuring the *compatibility* among snapshots is crucial. To this end, it is practice to formulate and solve the differential problem over a fixed, *parameter-independent* domain $\Omega \subset \mathbb{R}^d$ [37]. This can be accomplished by introducing a parametrized map $\boldsymbol{\Phi} : \Omega \times \mathcal{P}_g \to \widetilde{\Omega}$ such that

$$\widetilde{\Omega}(\boldsymbol{\mu}_g) = \boldsymbol{\Phi}(\Omega;\,\boldsymbol{\mu}_g)\,.$$

The transformation $\boldsymbol{\Phi}(\cdot;\,\boldsymbol{\mu}_g)$ allows one to restate the general problem (2.1). Let V be a suitable Hilbert space over Ω and V' be its dual. Suppose V is equipped with the scalar product $(\cdot,\cdot)_V$ and the induced norm $\|\cdot\|_V = \sqrt{(\cdot,\cdot)_V}$. Given the parametrized map $G : V \times \mathcal{P} \to V'$ representing the (nonlinear) PDE over the reference domain Ω, we focus on problems of the form: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u(\boldsymbol{\mu}) \in V$ such that

$$G(u(\boldsymbol{\mu});\,\boldsymbol{\mu}) = 0 \quad \text{in } V'\,. \tag{2.2}$$

The weak formulation of problem (2.2) reads: given $\boldsymbol{\mu} \in \mathcal{P}$, seek $u(\boldsymbol{\mu}) \in V$ such that

$$g(u(\boldsymbol{\mu}),\,v;\,\boldsymbol{\mu}) = 0 \quad \forall v \in V\,, \tag{2.3}$$

where $g : V \times V \times \mathcal{P} \to \mathbb{R}$ is defined as

$$g(w,\,v;\,\boldsymbol{\mu}) = \langle G(w;\,\boldsymbol{\mu}),\,v\rangle_{V',V} \quad \forall w,\,v \in V\,,\ \forall \boldsymbol{\mu} \in \mathcal{P}\,,$$

with $\langle\cdot,\,\cdot\rangle_{V',V} : V' \times V \to \mathbb{R}$ the dual pairing between V and V'. Observe that the explicit expression of $g(\cdot,\,\cdot;\,\boldsymbol{\mu})$ involves the map $\boldsymbol{\Phi}(\cdot;\,\boldsymbol{\mu}_g)$, thus keeping track of the original domain $\widetilde{\Omega}(\boldsymbol{\mu}_g)$. Then, the solution $\widetilde{u}(\boldsymbol{\mu})$ over the original domain $\widetilde{\Omega}(\boldsymbol{\mu}_g)$ can be recovered as

$$\widetilde{u}(\boldsymbol{\mu}) = u(\boldsymbol{\mu}) \circ \boldsymbol{\Phi}(\boldsymbol{\mu}_g)\,.$$

In our numerical tests, we employ the squared reference domain shown on the right in Fig. 5 and we resort to a particular choice for $\boldsymbol{\Phi}(\cdot;\,\boldsymbol{\mu}_g)$ – the boundary displacement-dependent transfinite map (BDD TM) proposed by Jaggli et al. [26].

2.2. Discrete full-order model

Let $V_h \subset V$ be a suitable FE subspace of V of (finite) dimension N_h, with $h \geq 0$ being the characteristic size of the underlying mesh Ω_h which discretizes Ω. The FE approximation of the weak problem (2.3) can be cast in the form: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u_h(\boldsymbol{\mu}) \in V_h$ such that

$$g(u_h(\boldsymbol{\mu}),\,v_h;\,\boldsymbol{\mu}) = 0 \quad \forall v_h \in V_h\,. \tag{2.4}$$

From an algebraic standpoint, letting $\{\phi_1,\,\dots,\,\phi_{N_h}\}$ be a Lagrangian basis for V_h and denoting by $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^{N_h}$ the vector collecting the *nodal values* $\{u_h^{(1)}(\boldsymbol{\mu}),\,\dots,\,u_h^{(N_h)}(\boldsymbol{\mu})\}$ of $u_h(\boldsymbol{\mu})$, problem (2.4) is equivalent to: given $\boldsymbol{\mu} \in \mathcal{P}$, find $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^{N_h}$ such that

$$\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\,\boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^{N_h}\,, \tag{2.5}$$

where the i-th component of the *residual vector* $\mathbf{G}_h(\cdot;\,\boldsymbol{\mu})$ is given by

$$\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\,\boldsymbol{\mu})\big)_i = g(u_h(\boldsymbol{\mu}),\,\phi_i;\,\boldsymbol{\mu})\,, \quad i = 1,\,\dots,\,N_h\,. \tag{2.6}$$

Observe that, due to the nonlinearity of $g(\cdot,\,\cdot;\,\boldsymbol{\mu})$ in its first argument, one has to resort to some iterative method, e.g., Newton's method, to solve the *Galerkin* problem (2.5).

3. Projection-based reduced basis method

As outlined above, the finite element discretization of the $\boldsymbol{\mu}$-dependent nonlinear differential problem (2.3), combined with Newton's method, entails the assembly and solution of (possibly) many linear systems, whose dimension N_h is directly related to (*i*) the size of the underlying grid and (*ii*) the order of the polynomial FE space adopted. Since the accuracy of the resulting discretization heavily relies on these two factors, a direct numerical approximation of the full-order model implies severe computational costs. Therefore, this approach is not affordable in *many-query* and *real-time* contexts. This motivates the use of reduced order models. Particularly, reduced basis methods seek an approximate solution to problem (2.3) as a linear combination of parameter-independent functions $\{\psi_1, \dots, \psi_L\} \subset V_h$, called *reduced basis functions*, built from a collection of high-fidelity snapshots $\{u_h(\boldsymbol{\mu}^{(1)}), \dots, u_h(\boldsymbol{\mu}^{(N)})\}$, where the discrete and finite set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathcal{P}$ may consist of either a uniform lattice or randomly generated points over the parameter domain $\mathcal{P}$ [23]. The basis functions $\{\psi_l\}_{1 \le l \le L}$ generally follow from a principal component analysis (PCA) of the set of snapshots (in that case, $N > L$), or they might coincide with the snapshots themselves (in that case, $N = L$). In the latter approach, typical of any *greedy* method, the parameters $\{\boldsymbol{\mu}^{(n)}\}_{1 \le n \le N}$ must be carefully chosen following some optimality criterium (see, e.g., [11]). Here, we pursue the first approach, employing the well-known proper orthogonal decomposition (POD) method [32,49], detailed in the following subsection. For now, assume that a reduced basis is available and let $V_{\mathtt{rb}} \subset V_h$ be the associated *reduced basis space*, i.e.,

$$V_{\mathtt{rb}} = \text{span}\{\psi_1, \dots, \psi_L\}.$$

A *reduced basis solution* $u_L(\boldsymbol{\mu})$ is sought in the form

$$u_L(\boldsymbol{x};\, \boldsymbol{\mu}) = \sum_{l=1}^{L} u_{\mathtt{rb}}^{(l)}(\boldsymbol{\mu})\, \psi_l(\boldsymbol{x}) \;\in\; V_{\mathtt{rb}},$$

with

$$\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) = \left[u_{\mathtt{rb}}^{(1)}(\boldsymbol{\mu}), \dots, u_{\mathtt{rb}}^{(L)}(\boldsymbol{\mu})\right]^T \in \mathbb{R}^L$$

being the *reduced coefficients* (also called *generalized coordinates*) for the expansion of the RB solution in the RB basis functions. To recover $u_L(\boldsymbol{\mu})$, we proceed to project the variational problem (2.3) onto the RB space $V_{\mathtt{rb}}$ by pursuing a standard Galerkin approach, leading to the following *reduced basis problem*: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u_L(\boldsymbol{\mu}) \in V_{\mathtt{rb}}$ so that

$$g(u_L(\boldsymbol{\mu}),\, v_L;\, \boldsymbol{\mu}) = 0 \quad \forall v_L \in V_{\mathtt{rb}}. \tag{3.1}$$

Denoting by $\boldsymbol{\psi}_l \in \mathbb{R}^{N_h}$ the vector gathering the nodal values of ψ_l, for $l = 1, \dots, L$, let us introduce the matrix

$$\mathbb{V} = \left[\boldsymbol{\psi}_1 \,|\, \dots \,|\, \boldsymbol{\psi}_L\right] \in \mathbb{R}^{N_h \times L}.$$

For any $v_L \in V_{\mathtt{rb}}$, $\mathbb{V}$ encodes the change of variables from the RB basis to the Lagrangian FE basis, i.e.,

$$\mathbf{v}_L = \mathbb{V}\, \mathbf{v}_{\mathtt{rb}}. \tag{3.2}$$

Then, due to (2.6) and (3.2), the algebraic formulation of the reduced basis problem (3.1) reads: given $\boldsymbol{\mu} \in \mathcal{P}$, seek $\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^L$ such that

$$\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu});\, \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L(\boldsymbol{\mu});\, \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbb{V}\, \mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu});\, \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L. \tag{3.3}$$

3.1. Proper orthogonal decomposition

Consider a collection of N snapshots $\{u_h(\boldsymbol{\mu}^{(1)}), \dots, u_h(\boldsymbol{\mu}^{(N)})\}$, corresponding to the finite and discrete parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathcal{P}$, and let $\mathcal{M}_{\Xi_N}$ be the associated subspace, i.e.,

$$\mathcal{M}_{\Xi_N} = \text{span}\{u_h(\boldsymbol{\mu}^{(1)}), \dots, u_h(\boldsymbol{\mu}^{(N)})\}.$$

We assume that $\mathcal{M}_{\Xi_N}$ provides a good approximation of the *discrete solution manifold* $\mathcal{M}_h$,

$$\mathcal{M}_h = \{u_h(\boldsymbol{\mu}) \,:\, \boldsymbol{\mu} \in \mathcal{P}\},$$

as long as the number of snapshots is sufficiently large (but typically much smaller than the dimension N_h of the FE space). Then, we aim at finding a parameter-independent *reduced basis* for $\mathcal{M}_{\Xi_N}$, i.e., a collection of FE functions $\{\psi_1, \dots, \psi_L\} \subset \mathcal{M}_{\Xi_N}$, with $L \ll N_h$ and L *independent* of N_h, so that the associated linear space constitutes a low-rank approximation of

$\mathcal{M}_{\Xi_N}$, optimal in some sense to be defined later. To this end, consider the *snapshot matrix* $\mathbb{S} \in \mathbb{R}^{N_h \times N}$ gathering the nodal values of the snapshots in a column-wise sense, i.e.,

$$\mathbb{S} = \left[\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \,\middle|\, \dots \,\middle|\, \mathbf{u}_h(\boldsymbol{\mu}^{(N)}) \right].$$

Denoting by R the rank of $\mathbb{S}$, with $R \leq \min\{N_h, N\}$, the singular value decomposition (SVD) of $\mathbb{S}$ ensures the existence of two orthogonal matrices $\mathbb{W} = \left[\mathbf{w}_1 \,\middle|\, \dots \,\middle|\, \mathbf{w}_{N_h} \right] \in \mathbb{R}^{N_h \times N_h}$ and $\mathbb{Z} = \left[\mathbf{z}_1 \,\middle|\, \dots \,\middle|\, \mathbf{z}_N \right] \in \mathbb{R}^{N \times N}$, and a diagonal matrix $\mathbb{D} = \mathrm{diag}(\sigma_1, \dots, \sigma_R) \in \mathbb{R}^{R \times R}$, with $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$, such that

$$\mathbb{S} = \mathbb{W} \begin{bmatrix} \mathbb{D} & 0 \\ 0 & 0 \end{bmatrix} \mathbb{Z}^T = \mathbb{W} \, \Sigma \, \mathbb{Z}^T,$$

where the zeros denote null matrices of appropriate dimensions. The real values $\{\sigma_i\}_{1 \leq i \leq R}$ are called *singular values* of $\mathbb{S}$, the columns $\{\mathbf{w}_m\}_{1 \leq m \leq N_h}$ of $\mathbb{W}$ are called *left singular vectors* of $\mathbb{S}$, and the columns $\{\mathbf{z}_n\}_{1 \leq n \leq N}$ of $\mathbb{Z}$ are called *right singular vectors* of $\mathbb{S}$, and they are related by the following relations:

$$\begin{aligned} &\mathbb{S}\mathbb{S}^T \mathbf{w}_m = \begin{cases} \sigma_m^2 \, \mathbf{w}_m & \text{for } 1 \leq m \leq R, \\ \mathbf{0} & \text{for } R+1 \leq m \leq N_h, \end{cases} \qquad \mathbb{S}^T \mathbb{S} \, \mathbf{z}_n = \begin{cases} \sigma_n^2 \, \mathbf{z}_n & \text{for } 1 \leq n \leq R, \\ \mathbf{0} & \text{for } R+1 \leq n \leq N, \end{cases} \\ &\mathbb{S} \, \mathbf{z}_i = \sigma_i \, \mathbf{w}_i \quad \text{for } 1 \leq i \leq R, \qquad \mathbb{S}^T \mathbf{w}_i = \sigma_i \, \mathbf{z}_i \quad \text{for } 1 \leq i \leq R. \end{aligned} \tag{3.4}$$

At the algebraic level, our goal is to approximate the columns of $\mathbb{S}$ by means of L orthonormal vectors $\{\widetilde{\mathbf{w}}_1, \dots, \widetilde{\mathbf{w}}_L\}$, with $L < R$. It is an easy matter to show that for each $\mathbf{s}_n$, $n = 1, \dots, N$, the element of $\mathrm{span}\{\widetilde{\mathbf{w}}_1, \dots, \widetilde{\mathbf{w}}_L\}$ closest to $\mathbf{s}_n$ in the Euclidean norm $\|\cdot\|_{\mathbb{R}^{N_h}} = \sqrt{(\cdot,\cdot)_{\mathbb{R}^{N_h}}}$ is given by

$$\sum_{l=1}^{L} \left(\mathbf{s}_n, \widetilde{\mathbf{w}}_l \right)_{\mathbb{R}^{N_h}} \widetilde{\mathbf{w}}_l.$$

Hence, we could measure the error committed by approximating the columns of $\mathbb{S}$ via the vectors $\{\widetilde{\mathbf{w}}_l\}_{1 \leq l \leq L}$ through the quantity

$$\varepsilon(\widetilde{\mathbf{w}}_1, \dots, \widetilde{\mathbf{w}}_L) = \sum_{n=1}^{N} \left\| \mathbf{s}_n - \sum_{l=1}^{L} \left(\mathbf{s}_n, \widetilde{\mathbf{w}}_l \right)_{\mathbb{R}^{N_h}} \widetilde{\mathbf{w}}_l \right\|_{\mathbb{R}^{N_h}}^2. \tag{3.5}$$

The Schmidt–Eckart–Young theorem [17,47] states that the *POD basis* of rank L $\{\mathbf{w}_1, \dots, \mathbf{w}_L\}$, consisting of the first L left singular values of $\mathbb{S}$, minimizes (3.5) among all the orthonormal bases of $\mathbb{R}^L$. Therefore, in the POD-Galerkin RB method, we set $\boldsymbol{\psi}_l = \mathbf{w}_l$, for all $l = 1, \dots, L$, so that

$$\mathbb{V} = \left[\mathbf{w}_1 \,\middle|\, \dots \,\middle|\, \mathbf{w}_L \right].$$

From a computational viewpoint, the first L left singular vectors $\{\mathbf{w}_l\}_{1 \leq l \leq L}$ of $\mathbb{S}$ can be efficiently computed through the so-called *method of snapshots*. We should distinguish two cases:

(a) if $N_h \leq N$: directly solve the eigenvalue problems $\mathbb{S}\mathbb{S}^T \mathbf{w}_l = \lambda_l \, \mathbf{w}_l$, for $1 \leq l \leq L$;
(b) if $N_h > N$: compute the *correlation* matrix $\mathbb{M} = \mathbb{S}^T \mathbb{S}$ and solve the eigenvalue problems $\mathbb{M} \, \mathbf{z}_l = \lambda_l \, \mathbf{z}_l$, for $1 \leq l \leq L$. Then, by (3.4), set $\mathbf{w}_l = (\lambda_l)^{-1/2} \, \mathbb{S} \, \mathbf{z}_l$, for $1 \leq l \leq L$.

3.2. Implementation: details and issues

The numerical procedure presented so far can be efficiently carried out within an offline-online framework [42]. The parameter-independent *offline* step consists of the generation of the snapshots through a high-fidelity, expensive scheme and the subsequent construction of the reduced basis via POD. To determine an appropriate dimension for the basis, which ensures a desired degree of accuracy, one can resort to empirical criteria, like, e.g., the *relative information content* [44]. Then, given a new parameter value $\boldsymbol{\mu} \in \mathcal{P}$, the nonlinear reduced system (3.3) is solved *online*.

However, to enjoy a significant reduction in the computational burden with respect to traditional (full-order) discretization techniques, the complexity of any online query should be *independent* of the original size of the problem. Yet, due to the nonlinearity of the underlying PDE and the non-affinity in the parameter dependence (partially induced by the transformation map $\boldsymbol{\Phi}(\cdot; \boldsymbol{\mu}_g)$), the assembly of the reduced problems has to be embodied directly in the online stage, thus seriously compromising the efficiency of the overall procedure [3]. Without escaping the algebraic framework, this can be overcome by resorting to suitable techniques such as the discrete empirical interpolation method (DEIM) [10] or its matrix variant (MDEIM) [40], aiming at recovering an affine dependency on the parameter $\boldsymbol{\mu}$. However, the implementation of

such techniques is problem-dependent and of an *intrusive* nature, as it requires one to modify the assembly routines of the corresponding computational code [9]. Moreover, any interpolation procedure unavoidably introduces a further level of approximation. As a matter of fact, typically one needs to generate a larger number of snapshots in the offline stage and then retain a larger number of POD modes to guarantee the same accuracy provided by the standard POD-Galerkin method [3].

4. Artificial neural networks

Inspired by the biological information processing system (see, e.g., [22,28]), an *artificial neural network* (ANN), often referred to as *neural network*, is a computational model able to learn from observational data, i.e., by example, thus providing an alternative to the algorithmic programming paradigm [39]. As its original counterpart, it consists of a collection of processing units, called (artificial) *neurons*, and a set of directed *weighted synaptic* connections among the neurons. Data travel among neurons through the connections, following the direction imposed by the synapses. Hence, an artificial neural network is an *oriented graph*, with the neurons as *nodes* and the synapses as oriented *edges*, whose weights are adjusted by means of a *training* process to configure the network for a specific application [48].

In the following, we discuss the structure and training of a neural network, starting by detailing the working principles of an artificial neuron.

4.1. Neuronal model

An artificial neuron represents a simplified model of a biological neuron [28]. To introduce the components of the model, let us consider the neuron j represented on the left in Fig. 1. Suppose that it is connected with m sending neurons $\{s_1, \dots, s_m\}$, and n receiving (target) neurons $\{r_1, \dots, r_n\}$. Denoting by $y_\alpha(t) \in \mathbb{R}$ the (scalar) output of a generic neuron α at time t and by $w_{\alpha,\beta}$ the weight of the connection (α, β), neuron j gets the weighted inputs $w_{s_k,j}\, y_{s_k}(t)$, $k = 1, \dots, m$, at time t, and sends out the output $y_j(t + \Delta t)$ to the target neurons $\{r_1, \dots, r_n\}$ at time $t + \Delta t$. In particular, neuron r_i, $i = 1, \dots, n$, receives as input $w_{j,r_i}\, y_j(t + \Delta t)$. Note that in the context of ANNs, time is discretized by introducing the timestep Δt. This is clearly not plausible from a biological viewpoint; however, it substantially simplifies the implementation. In the following, we will avoid to specify the dependence on time unless strictly necessary, to lighten the notation.

An artificial neuron j is completely characterized by three functions: the propagation function, the activation function and the output function. The *propagation function* f_{prop} converts the vectorial input $\mathbf{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ into a scalar u_j often called *net input*, i.e.,

$$u_j = f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) .$$

A common choice for f_{prop} (used also in this work) is the weighted sum, adding up the scalar inputs multiplied by their respective weights:

$$f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) = \sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} .$$

At each timestep, the *activation state* a_j, often referred to as *activation*, quantifies to which extent neuron j is currently *active* or *excited*. It results from the *activation function* f_{act}, which combines the net input u_j with a threshold $\theta_j \in \mathbb{R}$ [28]:

$$a_j = f_{act}(u_j; \theta_j) = f_{act}\Big(\sum_{k=1}^{m} w_{s_k,j}\, y_{s_k}\,; \theta_j\Big) .$$

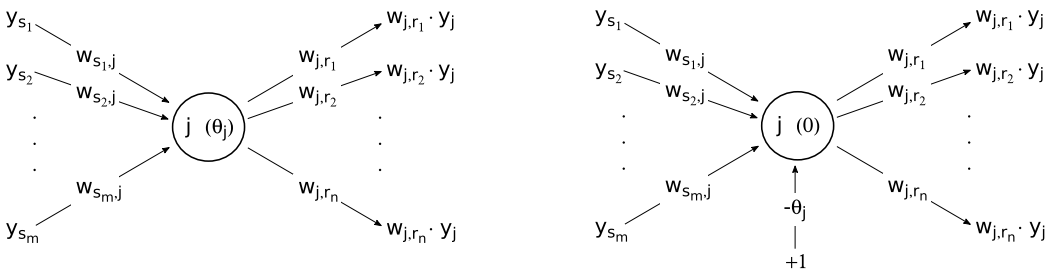


Fig. 1. Visualization of the generic j-th neuron of an artificial neural network, including (right) or not (left) a bias neuron. On the left, the neuron accumulates the weighted inputs $\{w_{s_1,j}\, y_{s_1}, \dots, w_{s_m,j}\, y_{s_m}\}$ respectively coming from the sending neurons $\{s_1, \dots, s_m\}$; on the right, the neuron accumulates the weighted inputs $\{w_{s_1,j}\, y_{s_1}, \dots, w_{s_m,j}\, y_{s_m}, -\theta_j\}$ respectively coming from the sending neurons $\{s_1, \dots, s_m, b\}$, with b the bias neuron. In both situations, the neuron then fires y_j, sent to the target neurons $\{r_1, \dots, r_n\}$ through the synapsis $\{w_{j,r_1}, \dots, w_{j,r_n}\}$. The neuron threshold is reported in brackets within its body.

Note that the threshold θ_j is a parameter of the network and as such one may choose to adapt it through a training process, exactly as it can be done for the synaptic weights. To ease the runtime access of θ_j, it is common practice to introduce a *bias neuron* in the network. A bias neuron is a continuously firing neuron, with constant output $y_b = 1$, which is directly connected with neuron j, assigning the *bias weight* $w_{b,j} = -\theta_j$ to the connection. As can be deduced by the representation on the right in Fig. 1, θ_j is now treated as a synaptic weight, while the neuron threshold is set to zero. Therefore, the net input and the activation state can respectively be expressed as

$$u_j = \sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} - \theta_j \quad \text{and} \quad a_j = f_{act}\Big(\sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} - \theta_j\Big)\,.$$

There exist various choices for the activation function. The sigmoid activation functions have been widely used for the realization of artificial neural networks due to their graceful combination of linear and nonlinear behavior [22]. Sigmoid functions are s-shaped, monotonically increasing, and assume values in a bounded interval; a well-known instance is given by the hyperbolic tangent,

$$f_{act}(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}\,.$$

Finally, the *output function* f_{out} calculates the scalar *output* $y_j \in \mathbb{R}$ based on the activation state a_j of the neuron:

$$y_j = f_{out}(a_j)\,.$$

Often, f_{out} is the identity function, so that activation and output of a neuron coincides, i.e., $y_j = f_{out}(a_j) = a_j$. The output y_j could then be sent either to other neurons or constitute a component of the overall output vector of the network, as for the neurons in the output layer of a feedforward neural network, illustrated in the following subsection.

It should be pointed out that the neural model presented so far refers to the so called *computing* neuron, i.e., a neuron which processes input information to provide a response. However, in a neural network one may also identify *source* neurons, supplying the network with the respective components of the input vector, without performing any computation [22].

4.2. Network topology: the feedforward neural network

The interconnection of neurons within a network defines the *topology* of the network itself, i.e., its design. In the literature, many network architectures have been proposed, sometimes tailored to a specific application. Among all, *feedforward neural networks*, also called *perceptrons* [46], have been preferred in function regression tasks.

In a feedforward neural network, neurons are arranged into *layers*, with one *input layer* of M_I source neurons, K *hidden layers*, each one consisting of H_k computing neurons, $k = 1, \dots, K$, and an *output layer* of M_O computing neurons. As a characteristic property, neurons in a layer can only be connected with neurons in the next layer towards the output layer. When an input vector $\mathbf{p} \in \mathbb{R}^{M_I}$ is supplied to the network through the source nodes, this provides the input signal for the neurons in the first hidden layer. In turn, the outputs of each hidden layer feed the neurons in the following layer. In this way, information travels towards the output layer, whose outputs constitute the components of the overall output $\mathbf{q} \in \mathbb{R}^{M_O}$ of the network.[1] Hence, a feedforward network establishes a map between the *input space* $\mathbb{R}^{M_I}$ and the *output space* $\mathbb{R}^{M_O}$. This does make this network architecture particularly suitable for continuous function approximation.

Feedforward networks can be classified according to the number of hidden layers or, equivalently, the number of layers of trainable weights. Single-layer perceptrons (SLPs) consist of the input and output layer, without any hidden layer. Because of their simple structure, the range of application of SLPs is rather limited. Indeed, only *linearly separable* data can be properly represented using SLPs [28]. Conversely, multi-layer perceptrons (MLPs), with at least one hidden layer, are universal function approximators, as stated by Cybenko [12,13]. In detail:

(i) MLPs with *one* hidden layer and differentiable activation functions can approximate *any continuous* function;
(ii) MLPs with *two* hidden layers and differentiable activation functions can approximate *any* function.

Therefore, in many practical applications there is no reason to employ MLPs with more than two hidden layers. However, (i) and (ii) do not give any practical advice neither on the number of hidden neurons nor the number of samples required to train the network: these should be found pursuing a *trial-and-error* (and likely time-consuming) approach [21].

An instance of a three-layer (i.e., two hidden layer plus the output layer) feedforward network is offered in Fig. 2. In this case, we have $M_I = 3$ input neurons (denoted with the letter i), $H_1 = H_2 = 6$ hidden neurons (letter h for both hidden layers), and $M_O = 4$ output neurons (letter o). In particular, it represents an instance of a *completely linked* perceptron, since each neuron is directly connected with all neurons in the following layer.

[1] Please note that while the output of a single neuron is denoted with the letter y, we use the letter $\mathbf{q}$ (bolded) to indicate the overall output of the network. Clearly, for the j-th output neuron, its output y_j coincides with the corresponding entry of $\mathbf{q}$, i.e., $q_j = y_j$ for any $j = 1, \dots, M_O$.

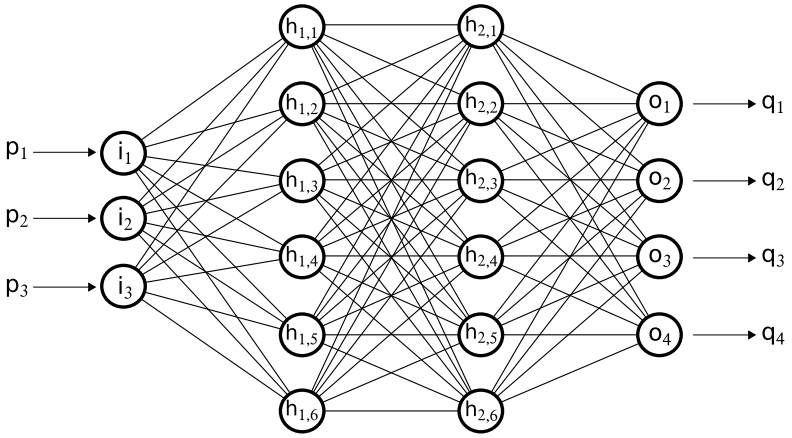


Fig. 2. A three-layer feedforward neural network, with three input neurons, two hidden layers each one consisting of six neurons, and four output neurons. Within each connection, information flows from left to right.

4.3. Training a multi-layer feedforward neural network

As previously mentioned, the principal characteristic of a neural network is its capability of *learning* from the surrounding environment, storing the acquired knowledge via its internal parameters, i.e., the synaptic and bias weights. Learning is accomplished through a training process, during which the network is exposed to a collection of examples, called *training data*. According to some performance measure, the weights are then adjusted by means of a well-defined set of rules. Therefore, the learning procedure is an *algorithm*, typically iterative, such that after a successful training, the neural network provides reasonable responses for unknown problems of the same class of the training set. This property is known as *generalization* [28].

Training algorithms can be classified based on the nature of the training set, i.e., the set of training data. We can then distinguish three *learning paradigms*: supervised learning, unsupervised learning and reinforcement learning [21]. The choice of the learning paradigm is clearly task-dependent. Particularly, for function approximation, the *supervised* learning paradigm is the natural choice.

Consider the nonlinear unknown function $\boldsymbol{f} : \mathbb{R}^{M_I} \to \mathbb{R}^{M_O}$ and a set of labeled examples $\left\{\mathbf{p}_i, \mathbf{t}_i = \boldsymbol{f}(\mathbf{p}_i)\right\}_{1 \le i \le N_{tr}}$, which form the training set. Note that to each *training input* $\mathbf{p}_i \in \mathbb{R}^{M_I}$, $i = 1, \dots, N_{tr}$, corresponds an associated *desired output* $\mathbf{t}_i \in \mathbb{R}^{M_O}$. The goal is to approximate $\boldsymbol{f}$ over a domain $D \subset \mathbb{R}^{M_I}$ up to a user-defined tolerance ϵ, i.e.,

$$||\boldsymbol{F}(\mathbf{x}) - \boldsymbol{f}(\mathbf{x})|| < \epsilon \quad \forall \mathbf{x} \in D\,,$$

where $\boldsymbol{F} : \mathbb{R}^{M_I} \to \mathbb{R}^{M_O}$ is the actual input-output map established by the neural network and $||\cdot||$ is some suitable norm on $\mathbb{R}^{M_O}$. For this purpose, consider the synapsis between a sending neuron i and a target neuron j. At the t-th iteration (also called *epoch*) of the training procedure, the weight $w_{i,j}(t)$ of the connection (i, j) is modified by the time-dependent quantity $\Delta w_{i,j}(t)$, whose form depends on the specific learning rule. Hence, at the subsequent iteration $t + 1$ the synaptic weight is simply given by

$$w_{i,j}(t+1) = w_{i,j}(t) + \Delta w_{i,j}(t)\,.$$

The whole training process is driven by an *error* or *performance* function E, which measures the discrepancy between the neural network knowledge of the surrounding environment and the actual state of the environment itself. Therefore, every learning rule aims to *minimize* the performance E, thought as a scalar function of the free parameters of the network, namely

$$E = E(\mathbf{w}) \in \mathbb{R}\,,$$

with $\mathbf{w} \in \mathbb{R}^W$ being the vector collecting all the synaptic and bias weights. Thus, the point over the error surface reached at the end of a successful training process provides the *optimal* configuration $\mathbf{w}_{opt}$ for the network. A common choice for the performance function is the (accumulated) mean squared error (MSE)

$$E(\mathbf{w}) = \sum_{\mathbf{p} \in P} E_{\mathbf{p}}(\mathbf{w}) = \sum_{\mathbf{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\mathbf{p},j} - q_{\mathbf{p},j}\right)^2\,, \tag{4.1}$$

where for each input vector $\mathbf{p}$ belonging to the training set P, $\mathbf{t_p}$ and $\mathbf{q_p}$ denote respectively the corresponding desired and actual output. Observe that (4.1) accounts for the error committed on each input vector in the training set, leading to an *offline* learning procedure.

In our numerical tests, the weights are iteratively adjusted via the Levenberg–Marquardt (LM) algorithm [30,34]. With respect to the notation used in [20], in our numerical tests we set $\mu = 0.001$ as starting point for the bias parameter μ, with the damping factor β equal to 10. As pointed out in [21], the cost of the LM algorithm increases nonlinearly with the number of neurons in the network, making it poorly efficient for large networks. However, it turns out to be efficient and accurate for networks with few hundreds of connections.

5. A non-intrusive reduced basis method using artificial neural networks

The scenario portrayed so far motivates the research for an alternative approach to tackle any online query within the reduced basis framework. To this end, let us remark that there exists a one-to-one correspondence between the reduced space V_{rb} and the column space $\mathrm{Col}(\mathbb{V})$ of $\mathbb{V}$. Indeed, letting $\{\phi_1, \dots, \phi_{N_h}\}$ be a basis for V_h and $\{\psi_1, \dots, \psi_L\}$ be the reduced basis, from Eq. (3.2) follows:

$$V_{\mathrm{rb}} \ni v_L = \sum_{j=1}^{L} v_{\mathrm{rb}}^{(j)}\, \psi_j = \sum_{j=1}^{L} v_{\mathrm{rb}}^{(j)} \sum_{i=1}^{N_h} \mathbb{V}_{i,j}\, \phi_i = \sum_{i=1}^{N_h} \left(\mathbb{V}\, \mathbf{v}_{\mathrm{rb}}\right)_i \phi_i \quad \leftrightarrow \quad \mathbf{v}_L \in \mathrm{Col}(\mathbb{V})\,.$$

In particular, this implies that the projection of any $v_h \in V_h$ onto V_{rb} in the discrete scalar product $(\cdot,\cdot)_h$,

$$(\chi_h,\, \xi_h)_h = \sum_{i=1}^{N_h} \chi_h^{(i)}\, \xi_h^{(i)} = (\boldsymbol{\chi}_h,\, \boldsymbol{\xi}_h)_{\mathbb{R}^{N_h}} \quad \forall \chi_h,\, \xi_h \in V_h\,, \tag{5.1}$$

algebraically corresponds to the projection $\mathbf{v}_h^{\mathbb{V}}$ of $\mathbf{v}_h$ onto $\mathrm{Col}(\mathbb{V})$ in the Euclidean scalar product, given by

$$\mathbf{v}_h^{\mathbb{V}} = \mathbb{P}\, \mathbf{v}_h \quad \text{with} \quad \mathbb{P} = \mathbb{V}\mathbb{V}^T \in \mathbb{R}^{N_h \times N_h}\,.$$

Note that $\mathbf{v}_h^{\mathbb{V}}$ is the element of $\mathrm{Col}(\mathbb{V})$ closest to $\mathbf{v}_h$ in the Euclidean norm, i.e.,

$$\left\| \mathbf{v}_h - \mathbf{v}_h^{\mathbb{V}} \right\|_{\mathbb{R}^{N_h}} = \inf_{\mathbf{w}_h \in \mathrm{Col}(\mathbb{V})} \|\mathbf{v}_h - \mathbf{w}_h\|_{\mathbb{R}^{N_h}}\,.$$

Therefore, the element of V_{rb} closest to the high-fidelity solution u_h in the discrete norm $\|\cdot\|_h = \sqrt{(\cdot,\cdot)_h}$ can be expressed as

$$u_h^{\mathbb{V}}(\boldsymbol{x};\, \boldsymbol{\mu}) = \sum_{j=1}^{N_h} \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu})\right)_j \phi_j(\boldsymbol{x}) = \sum_{i=1}^{L} \left(\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu})\right)_i \psi_i(\boldsymbol{x})\,.$$

Motivated by this last equality, once a reduced basis has been constructed (e.g., via POD of the snapshot matrix), we aim at approximating the function

$$\begin{aligned} \boldsymbol{\pi} \;:\; \mathcal{P} \subset \mathbb{R}^P \;&\to\; \mathbb{R}^L \\ \boldsymbol{\mu} \;&\mapsto\; \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu})\,, \end{aligned} \tag{5.2}$$

which maps each input vector parameter $\boldsymbol{\mu} \in \mathcal{P}$ to the coefficients $\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu})$ for the expansion of $u_h^{\mathbb{V}}$ in the reduced basis $\{\psi_i\}_{1 \le i \le L}$. Then, given a new parameter instance $\boldsymbol{\mu}$, the associated RB solution is simply given by the evaluation at $\boldsymbol{\mu}$ of the so-built approximation $\hat{\boldsymbol{\pi}}$ of $\boldsymbol{\pi}$, i.e.

$$\mathbf{u}_{\mathrm{rb}}(\boldsymbol{\mu}) = \hat{\boldsymbol{\pi}}(\boldsymbol{\mu})\,,$$

and, consequently,

$$\mathbf{u}_L(\boldsymbol{\mu}) = \mathbb{V}\, \hat{\boldsymbol{\pi}}(\boldsymbol{\mu})\,.$$

Note that, provided that the construction of $\hat{\boldsymbol{\pi}}$ is carried out within the offline stage, this approach leads to a *non-intrusive* RB method, enabling a complete decoupling between the online step and the underlying full-order model. Moreover, the accuracy of the resulting reduced solution uniquely relies on the quality of the reduced basis and the effectiveness of the approximation $\hat{\boldsymbol{\pi}}$ of the map $\boldsymbol{\pi}$, which we assume sufficiently smooth.

In the literature, different approaches for the *interpolation* of (5.2) have been developed, e.g., exploiting some geometrical considerations concerning the discrete solution manifold $\mathcal{M}_h$ [1], or employing radial basis functions [11]. In this work we resort to artificial neural networks for the *nonlinear regression* of the map $\boldsymbol{\pi}$, leading to the POD-NN RB method. As

Algorithm 5.1 The offline and online stages for the POD-NN RB method.

function $\left[\mathbb{V}, \mathbf{w}_{\mathrm{rb}}\right] = \text{PODNN\_OFFLINE}(\mathcal{P}, \Omega_h, N)$
  generate the parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$
  compute the high-fidelity solutions $\{\mathbf{u}_h(\boldsymbol{\mu}^{(1)}), \dots, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})\}$ via FE-Newton's method
  generate the POD basis functions $\{\mathbf{w}_1, \dots, \mathbf{w}_L\}$ via method of snapshots
  assemble $\mathbb{V} = [\mathbf{w}_1 \,|\, \dots \,|\, \mathbf{w}_L]$
  find an optimal network configuration $\mathbf{w}_{\mathrm{rb}}$ relying upon LHS and Levenberg–Marquardt algorithm
end function

function $\mathbf{u}_L^{\mathrm{NN}}(\boldsymbol{\mu}) = \text{PODNN\_ONLINE}(\boldsymbol{\mu}, \mathbb{V}, \mathbf{w}_{\mathrm{rb}})$
  evaluate the output $\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu})$ of the network for the input vector $\boldsymbol{\mu}$
  compute $\mathbf{u}_L^{\mathrm{NN}}(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu})$
end function

described in Subsection 4.3, any neural network is tailored to the particular application at hand by means of a preliminary *training* phase. Here, we are concerned with a function regression task, thus we straightforwardly adopt a *supervised learning* paradigm, training the perceptron via exposition to a collection of (known) input-output pairs

$$P_{tr} = \left\{\left(\boldsymbol{\mu}^{(i)}, \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}^{(i)})\right)\right\}_{1\le i\le N_{tr}}.$$

According to the notation and nomenclature previously introduced, for any $i = 1, \dots, N_{tr}$, $\mathbf{p}_i = \boldsymbol{\mu}^{(i)} \in \mathbb{R}^P$ represents the *training input* and $\mathbf{t}_i = \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}^{(i)}) \in \mathbb{R}^L$ the associated *desired output*. In this respect, note that the output vectors $\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}^{(i)})$, $i = 1, \dots, N_{tr}$, are generated through the FE solver. On the one hand, this ensures the reliability of the teaching data, given the assumed high-fidelity of the FE scheme. On the other hand, this also suggests to incorporate the learning phase of the perceptron in the offline step of the POD-NN RB method, as described in Algorithm 5.1. In doing so, we exploit the natural decoupling between the training and the evaluation of neural networks, thus fulfilling the necessary requirement to ensure online efficiency.

It should be pointed out that the design of an effective learning procedure may require a larger amount of snapshots than the generation of the reduced space does. Moreover, we mentioned in Subsection 4.2 the time-consuming yet unavoidable *trial-and-error* approach which one should pursue in the search for an optimal network topology. To this end, we propose an automatic procedure.

While Cybenko's theorems (see Subsection 4.2) allow one to consider perceptrons with no more than two hidden layers, no similar a priori and general results are available for the number H of neurons per layer.[2] Hence, given an initial amount N_{tr} of training samples (say $N_{tr} = 100$), we train the network for increasing values of H, stopping when overfitting of training data occurs, due to an excessive number of neurons. In case the best configuration, i.e., the one yielding the smallest error on a *test* data set, does not meet a desired level of accuracy, we generate a new set of snapshots, which will enlarge the current training set, and then proceed to re-train the network in different configurations. At this point, it is worth pointing out two issues.

(i) Once the training set has been enriched, we can limit ourselves to network configurations including a number of neurons *no smaller* than the current optimal network. Indeed, the error on the test data set is decreasing in the number of training data the neural network is exposed to during the learning phase, and the larger the number of neurons, the faster the decay.
(ii) In order to maximize the additional quantity of information available for the learning, we should ensure that the new training inputs, i.e., parameter values, do not overlap with the ones already present in the training parameter set. To this aim, we pursue an heuristic approach, employing, at each iteration, the Latin hypercube sampling [25], which provides a good compromise between randomness and even coverage of the parameter domain.

The procedure is then iterated until a satisfactory degree of accuracy and generalization is attained, or the available resources (i.e., computing power, running time and memory space) are exhausted. Therefore, the speed-up enabled at the *online* stage comes at the cost of an extended *offline* phase.

In our numerical tests, we rely on the mean squared error (4.1) as performance function driving the neural network training process. To motivate this choice, let

$$\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu}) \in \mathbb{R}^L$$

be the (actual) output provided by the network for a given input $\boldsymbol{\mu}$, and

$$\mathbf{u}_L^{\mathrm{NN}}(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu}) \in \mathrm{Col}(\mathbb{V}) \subset \mathbb{R}^{N_h}.$$

[2] Here we uniquely consider networks with the same number of neurons in both the first and the second hidden layer, but this is not required.

Then (omitting the dependence on the input vector for ease of notation):

$$MSE\left(\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}},\, \mathbb{V}^T\mathbf{u}_h\right) \propto \left\|\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}} - \mathbb{V}^T\mathbf{u}_h\right\|_{\mathbb{R}^L}^2 = \left\|\mathbf{u}_L^{\mathrm{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\right\|_{\mathbb{R}^{N_h}}^2 = \left\|u_L^{\mathrm{NN}} - u_h^{\mathbb{V}}\right\|_h^2,$$

where

$$u_L^{\mathrm{NN}}(\boldsymbol{x};\,\boldsymbol{\mu}) = \sum_{i=1}^{L} \left(\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu})\right)_i\, \psi_i(\boldsymbol{x}) \;\in\; V_{\mathrm{rb}}\,.$$

Therefore, for any training input $\boldsymbol{\mu}^{(i)}$, $i = 1, \ldots, N_{tr}$, by minimizing the MSE we minimize the distance (in the discrete norm $\|\cdot\|_h$) between the approximation provided by the neural network and the projection of the FE solution onto the reduced space V_{rb}. The proper *generalization* to other parameter instances, not included in the training set, is then ensured by the implementation of suitable techniques (e.g., early stopping [35], generalized cross validation [27]) aiming at preventing the network to *overfit* the training data.

6. Numerical results

In this section, we present the numerical results obtained via the FE, POD-G and POD-NN methods applied to parametrized full-Dirichlet boundary value problems (BVPs) for the one-dimensional and two-dimensional nonlinear Poisson equation and the two-dimensional incompressible steady Navier–Stokes equations. In the one-dimensional case we deal uniquely with physical parameters, whereas in two spatial dimensions we consider purely geometric parametrizations.

The two RB techniques considered in this work are compared both in terms of accuracy and performance during the online stage. For this, let $\|\cdot\|$ be the canonical (Euclidean) norm on $\mathbb{R}^{N_h}$. In the online phases of the POD-G and POD-NN methods, for a new parameter value $\boldsymbol{\mu} \in \mathcal{P}$ (not involved either in the generation of the POD basis nor in the identification of an optimal neural network), the following *relative* errors with respect to the high-fidelity solution $\mathbf{u}_h(\boldsymbol{\mu})$ are analyzed:

(a) the POD-G relative error,

$$\varepsilon_{\mathrm{PODG}}(L, \boldsymbol{\mu}) = \frac{\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_L(\boldsymbol{\mu})\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|} = \frac{\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\,\mathbf{u}_{\mathrm{rb}}(\boldsymbol{\mu})\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|}\,; \tag{6.1}$$

(b) the POD-NN relative error,

$$\varepsilon_{\mathrm{PODNN}}(L, \boldsymbol{\mu}) = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_L^{\mathrm{NN}}(\boldsymbol{\mu})\right\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|} = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\,\mathbf{u}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu})\right\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|}\,; \tag{6.2}$$

(c) the relative projection error, i.e., the error committed by approximating the high-fidelity solution with its projection (in the discrete scalar product $(\cdot,\cdot)_h$, see (5.1)) onto the reduced space V_{rb},

$$\varepsilon_{\mathbb{V}}(L, \boldsymbol{\mu}) = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_h^{\mathbb{V}}(\boldsymbol{\mu})\right\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|} = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\right\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|}\,. \tag{6.3}$$

Clearly, (6.3) provides a lower-bound for both (6.1) and (6.2).

The above errors are evaluated on a *test* parameter set $\Xi_{te} \subset \mathcal{P}$ consisting of N_{te} randomly picked samples. Given that a sufficiently large number of test values is chosen, statistics for $\varepsilon_{\mathrm{PODG}}(L,\cdot)$, $\varepsilon_{\mathrm{PODNN}}(L,\cdot)$ and $\varepsilon_{\mathbb{V}}(L,\cdot)$ on Ξ_{te} can be reasonably assumed independent of the particular choice made for Ξ_{te}, thus making any subsequent error analysis reliable. In particular, in our numerical studies we let N_{te} range from 50 up to 100, and we consider the average of the errors over the test data set, respectively denoted by

$$\bar{\varepsilon}_{\mathrm{PODG}} = \bar{\varepsilon}_{\mathrm{PODG}}(L) = \frac{\sum_{\boldsymbol{\mu}\in\Xi_{te}} \varepsilon_{\mathrm{PODG}}(L, \boldsymbol{\mu})}{N_{te}}\,, \qquad \bar{\varepsilon}_{\mathrm{PODNN}} = \bar{\varepsilon}_{\mathrm{PODNN}}(L) = \frac{\sum_{\boldsymbol{\mu}\in\Xi_{te}} \varepsilon_{\mathrm{PODNN}}(L, \boldsymbol{\mu})}{N_{te}}\,,$$
$$\bar{\varepsilon}_{\mathbb{V}} = \bar{\varepsilon}_{\mathbb{V}}(L) = \frac{\sum_{\boldsymbol{\mu}\in\Xi_{te}} \varepsilon_{\mathbb{V}}(L, \boldsymbol{\mu})}{N_{te}}\,.$$

In the training phase of neural networks, we rely upon the well-known cross-validation procedure [27]. While the training samples $\Xi_{tr} \subset \mathcal{P}$ are generated via successive Latin hypercube samplings (as explained in Section 5), the *validation* inputs $\Xi_{va} \subset \mathcal{P}$ are randomly picked through a Monte Carlo sampling, as done for Ξ_{te}. The ratio between the size of Ξ_{va} and Ξ_{tr} is set to 0.3. At the beginning of the learning process, the weights are randomly initialized using a standard Gaussian distribution. To limit the dependence of the results on the starting configuration, we pursue the *multiple restarts* approach; see, e.g., [28,35]. Typically, five restarts are performed per network topology. Within each restart, an early stopping technique is implemented to further prevent overfitting: training is immediately stopped when the validation error increases over K_{ea}

consecutive epochs [35]. Here, $K_{ea} = 6$ is used. To ease the research for an optimal network configuration and by exploiting Cybenko's results (see Subsection 4.2), we limit ourselves to three-layers neural networks, with the same number of neurons for both hidden layers. For each topology, the hyperbolic tangent is used as activation function for the hidden neurons. All the results presented in the following subsections have been obtained via a MATLAB® implementation of the FE, POD-G and POD-NN methods, with all the simulations carried out on a laptop equipped with a CPU Intel® Core™ i7 @ 2.50 GHz.

6.1. Nonlinear Poisson equation

Despite a rather simple form, the Poisson equation has proved itself to be effective to model steady phenomena occurring in, e.g., electromagnetism, heat transfer and underground flows [36]. We consider the following version of the parametrized Poisson equation for a state variable $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$:

$$\begin{cases} -\widetilde{\nabla} \cdot \left(\widetilde{k}(\widetilde{\boldsymbol{x}},\, \widetilde{u}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu})\right) = \widetilde{s}(\widetilde{\boldsymbol{x}};\, \boldsymbol{\mu}_{ph}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g)\,, \quad \text{(a)} \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{h}(\widetilde{\boldsymbol{\sigma}};\, \boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}_D\,, \quad \text{(b)} \\ \widetilde{k}(\widetilde{\boldsymbol{\sigma}},\, \widetilde{u}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\boldsymbol{n}} = 0 & \text{on } \widetilde{\Gamma}_N\,. \quad \text{(c)} \end{cases} \tag{6.4}$$

Here, for any $\boldsymbol{\mu}_g \in \mathcal{P}_g$, $\widetilde{\boldsymbol{x}}$ and $\widetilde{\boldsymbol{\sigma}}$ denote a generic point in $\widetilde{\Omega}$ and on $\widetilde{\Gamma}$, respectively, $\widetilde{\nabla}$ is the nabla operator with respect to $\widetilde{\boldsymbol{x}}$, $\widetilde{\boldsymbol{n}} = \widetilde{\boldsymbol{n}}(\widetilde{\boldsymbol{\sigma}})$ denotes the outward normal to $\widetilde{\Gamma}$ in $\widetilde{\boldsymbol{\sigma}}$, $\widetilde{k} : \widetilde{\Omega} \times \mathbb{R} \times \mathcal{P}_{ph} \to (0, \infty)$ is the diffusion coefficient, $\widetilde{s} : \widetilde{\Omega} \times \mathcal{P}_{ph} \to \mathbb{R}$ is the source term, and $\widetilde{h} : \widetilde{\Gamma}_D \times \mathcal{P}_{ph} \to \mathbb{R}$ encodes the Dirichlet boundary conditions. To ease the subsequent discussion, we limit the attention to homogeneous Neumann boundary constraints.

Let us fix $\boldsymbol{\mu} \in \mathcal{P}$ and set

$$\widetilde{V} = \widetilde{V}(\boldsymbol{\mu}_g) = H^1_{\widetilde{\Gamma}_D}\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right) = \left\{v \in H^1\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right) \,:\, v|_{\widetilde{\Gamma}_D} = 0\right\},$$

Multiplying (6.4)(a) by a *test* function $v \in \widetilde{V}$, integrating over $\widetilde{\Omega}$, and exploiting integration by parts on the left-hand side, yields:

$$\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(\widetilde{u}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) = \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph})\, v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g)\,, \tag{6.5}$$

where we have omitted the dependence on the space variable $\widetilde{\boldsymbol{x}}$ for ease of notation. For the integrals in Eq. (6.5) to be well-defined, we require, for any $\boldsymbol{\mu}_g \in \mathcal{P}_g$,

$$|\widetilde{k}(\widetilde{\boldsymbol{x}},\, r;\, \boldsymbol{\mu}_g)| < \infty \text{ for almost any (a.a.) } \widetilde{\boldsymbol{x}} \in \widetilde{\Omega}(\boldsymbol{\mu}_g),\, r \in \mathbb{R}\,, \quad \text{and} \quad \widetilde{s}(\boldsymbol{\mu}_{ph}) \in L^2\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right).$$

Let then $\widetilde{l} = \widetilde{l}(\boldsymbol{\mu}) \in H^1\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right)$ be a *lifting* function such that $\widetilde{l}(\boldsymbol{\mu})|_{\widetilde{\Gamma}_D} = \widetilde{h}(\boldsymbol{\mu}_{ph})$, with $\widetilde{h}(\boldsymbol{\mu}_{ph}) \in H^{1/2}\left(\widetilde{\Gamma}_D\right)$. We assume that such a function can be constructed, e.g., by interpolation of the boundary condition. Hence, upon defining

$$\begin{aligned} \widetilde{a}(w,\, v;\, \boldsymbol{\mu}) &:= \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + \widetilde{l}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} w \cdot \widetilde{\nabla} v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \\ &\quad + \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + \widetilde{l}(\boldsymbol{\mu});\, \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{l}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \quad \forall w,\, v \in \widetilde{V}\,, \\ \widetilde{f}(v;\, \boldsymbol{\mu}) &:= \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph})\, v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \quad \forall v \in \widetilde{V}\,, \end{aligned} \tag{6.6}$$

the weak formulation of problem (6.4) reads: given $\boldsymbol{\mu} \in \mathcal{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{a}(\widetilde{u}(\boldsymbol{\mu}),\, v;\, \boldsymbol{\mu}) = \widetilde{f}(v;\, \boldsymbol{\mu}) \quad \forall v \in \widetilde{V}(\boldsymbol{\mu}_g)\,, \tag{6.7}$$

Then, the weak solution of problem (6.4) is given by $\widetilde{u}(\boldsymbol{\mu}) + \widetilde{l}(\boldsymbol{\mu})$.

Let us now re-state the variational problem (6.7) onto the reference domain Ω. For this purpose, let Γ_D and Γ_N be the portions of the boundary $\Gamma = \partial\Omega$ on which we impose Dirichlet and Neumann boundary conditions, respectively. Moreover, we denote by $\mathbb{J}_{\boldsymbol{\Phi}}(\cdot;\, \boldsymbol{\mu}_g)$ the Jacobian of the parametrized map $\boldsymbol{\Phi}(\cdot;\, \boldsymbol{\mu}_g)$, with determinant $|\mathbb{J}_{\boldsymbol{\Phi}}(\cdot;\, \boldsymbol{\mu}_g)|$, and we set

$$k(\boldsymbol{x},\, \cdot;\, \boldsymbol{\mu}) = \widetilde{k}(\boldsymbol{\Phi}(\boldsymbol{x};\, \boldsymbol{\mu}_g),\, \cdot;\, \boldsymbol{\mu}_{ph}),\; s(\boldsymbol{x};\, \boldsymbol{\mu}) = \widetilde{s}(\boldsymbol{\Phi}(\boldsymbol{x};\, \boldsymbol{\mu}_g);\, \boldsymbol{\mu}_{ph}) \quad \text{and} \quad h(\boldsymbol{x};\, \boldsymbol{\mu}) = \widetilde{h}(\boldsymbol{\Phi}(\boldsymbol{x};\, \boldsymbol{\mu}_g);\, \boldsymbol{\mu}_{ph})\,.$$

Letting

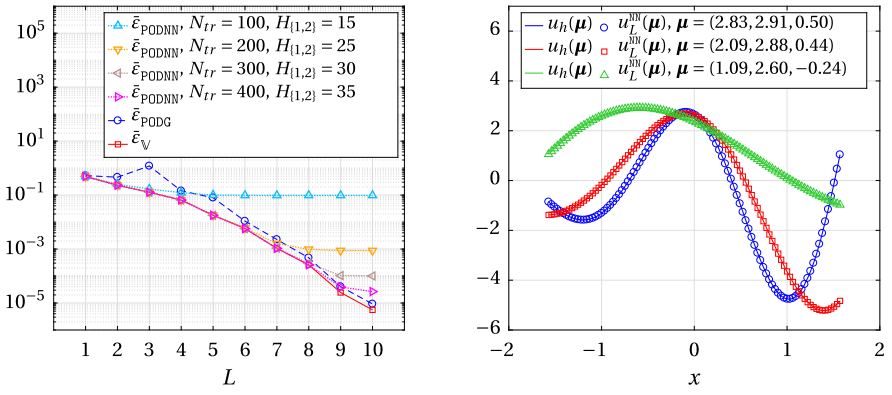


Fig. 3. Convergence analysis for the POD-G and POD-NN methods applied to problem (6.8) (*left*) and comparison between the FE and the POD-NN solutions for three parameter values (*right*). These latter results have been obtained via a neural network with $H_1 = H_2 = 35$ units per hidden layer and employing $L = 10$ POD modes.

$$V = H^1_{\Gamma_D}(\Omega)$$

and exploiting standard change of variables formulas, the variational formulation of the Poisson problem (6.4) over Ω reads: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u(\boldsymbol{\mu}) \in V$ such that

$$a(u(\boldsymbol{\mu}),\, v;\, \boldsymbol{\mu}) = f(v;\, \boldsymbol{\mu}) \quad \forall v \in V\,,$$

with

$$\begin{aligned} a(w,\, v;\, \boldsymbol{\mu}) &= \int_\Omega k(w + l(\boldsymbol{\mu});\, \boldsymbol{\mu})\, \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}_g) \nabla w \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}_g) \nabla v\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu}_g)|\, d\Omega \\ &\quad + \int_\Omega k(w + l(\boldsymbol{\mu});\, \boldsymbol{\mu})\, \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}_g) \nabla l(\boldsymbol{\mu}) \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}_g) \nabla v\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu}_g)|\, d\Omega\,, \\ f(v;\, \boldsymbol{\mu}) &= \int_\Omega s(\boldsymbol{\mu})\, v\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu}_g)|\, d\Omega\,, \end{aligned}$$

for any $w,\, v \in V$ and $\boldsymbol{\mu} \in \mathcal{P}$. Note that, as in (6.6), we resort to a lifting function $l(\boldsymbol{\mu}) \in H^1(\Omega)$ with $l(\boldsymbol{\mu})\big|_{\Gamma_D} = h(\boldsymbol{\mu})$, such that the weak solution to problem (6.4) re-stated over Ω is obtained as $u(\boldsymbol{\mu}) + l(\boldsymbol{\mu})$.

6.1.1. One-dimensional test case

Consider the following BVP for the one-dimensional Poisson equation, featuring an exponential nonlinearity (in the diffusion coefficient) and involving three parameters:

$$\begin{cases} -\big(\exp(u(\boldsymbol{\mu}))\, u(\boldsymbol{\mu})'\big)' = s(x;\, \boldsymbol{\mu}) \quad \text{in } \Omega = \left(-\dfrac{\pi}{2},\, \dfrac{\pi}{2}\right), \\ u(\boldsymbol{\mu})\big|_{x=\pm\pi/2} = \mu_2\, \sin\left(2 \pm \dfrac{\mu_1\, \pi}{2}\right) \exp\left(\pm \dfrac{\mu_3\, \pi}{2}\right). \end{cases} \tag{6.8}$$

Here, $\boldsymbol{\mu} = (\mu_1,\, \mu_2,\, \mu_3) \in \mathcal{P} = [1,\, 3] \times [1,\, 3] \times [-0.5,\, 0.5]$ and $s(\cdot;\, \boldsymbol{\mu})$ is defined such that

$$u_{ex}(x;\, \boldsymbol{\mu}) = \mu_2\, \sin\big(2 + \mu_1\, x\big)\, \exp\big(\mu_3\, x\big)$$

is the exact solution to the problem for any $\boldsymbol{\mu} \in \mathcal{P}$. The left plot of Fig. 3 shows the convergence of the POD-G and POD-NN solutions with respect to the number of basis functions L retained in the model, generated via POD of $N = 100$ FE snapshots computed over a uniform mesh of 100 nodes. Observe how the performance of the POD-NN method depends highly on the number of training data and hidden neurons used. In fact, for $N_{tr} = 100$ and $H_1 = H_2 = 15$ (upward-pointing triangles), the method can approximate only the first five generalized coordinates, actually providing more satisfactory results than POD-G does. Yet, the error stagnates for $L > 5$. However, as we expand the training set, we obtain more accurate predictions for a larger number of POD coefficients, provided that we allow the size of the network to increase. Particularly, employing $N_{tr} = 400$ training samples and $H_1 = H_2 = 35$ internal neurons (right-pointing triangles), the POD-NN error features an exponential decay for $L \leq 10$, resembling the projection error (squares). The reliability of the proposed RB method is also

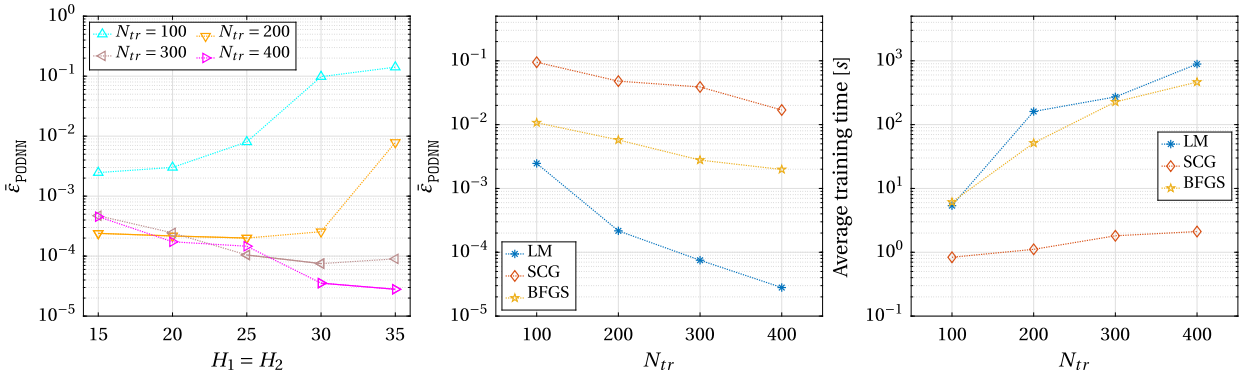


Fig. 4. *Left*: error analysis for the POD-NN RB method applied to problem (6.8). Several numbers of training samples are considered; the solid sections represent the steps followed by the automatic routine described in Section 5. *Center and right*: comparison between the LM, SCG and BFGS training algorithms.

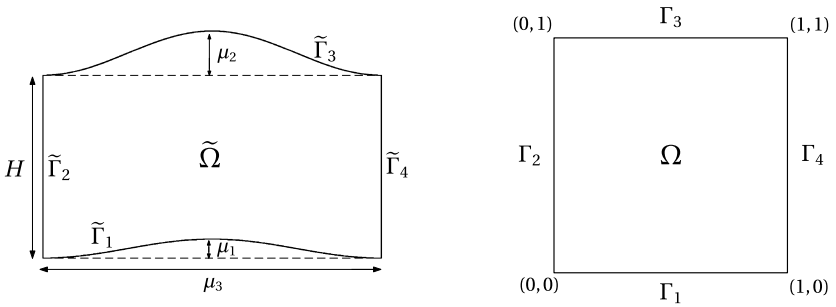


Fig. 5. The physical (*left*) and reference (*right*) domains for the Poisson problem (6.9).

confirmed by the right plot of Fig. 3, offering a comparison between the FE and POD-NN solutions for some parameter values.

Diving deeper into the analysis of the POD-NN method, the leftmost plot of Fig. 4 offers the results concerning the search for an optimal three-layers network configuration using the methodology described in Section 5. The steps followed by the routine are represented by solid tracts; however, for the sake of completeness, we also report the test error (i.e., the minimum over multiple restarts) for any considered number of hidden neurons per layer (H_1 and H_2) and any dimension of the training set (N_{tr}). We remark that the basic assumption which the proposed routine relies on is fulfilled: as the number of available samples increases, the number of neurons which allows to attain the minimum error increases as well, while the minimum error itself decreases.

To motivate its employment, in the rightmost panels of Fig. 4 we compare the LM algorithm against two other widely-used weight-updating routines: the scaled conjugate gradient (SCG) [38] and the BFGS quasi-Newton method [19]. The comparison takes into consideration the test error (*center*) and the average training time (*right*) for different amounts of training samples; the network architectures used are those suggested by the LM-based procedure analyzed above. For any considered size of the training set, LM turns out to be the most accurate method. For instance, for $N_{tr} = 400$ and $H_1 = H_2 = 35$, the average error yielded by LM is respectively two and three orders of magnitude smaller than that by BFGS and SCG. As one may expect, this comes with a cost: while BFGS requires 470 *s* and SCG only 1.1 *s*, LM averagely takes 900 *s*. However, we argue that the overheads associated with LM are well paid back by the large accuracy boost it guarantees. Further, recall that training is entirely performed *offline* and as such it does not affect the *online* running time. Yet, it is crucial to get reliable results. Hence, one should always sacrifice performance for the sake of accuracy during the offline phase, whenever resources are not constraining.

6.1.2. Two-dimensional test case

Let $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_3] \in \mathcal{P} = [-0.5, 0.5] \times [-0.5, 0.5] \times [1, 5]$ and $\widetilde{\Omega}(\boldsymbol{\mu})$ be the stenosis geometry reported on the left in Fig. 5, parametrized in the depths μ_1 and μ_2 of the bottom and top restrictions (or inflations), respectively, and the length μ_3 of the vessel. The Poisson problem we deal with reads:

$$\begin{cases} -\widetilde{\nabla} \cdot \left(\exp(\widetilde{u}(\boldsymbol{\mu}))\, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \right) = \widetilde{s}(\widetilde{x}, \widetilde{y}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{\sigma}_x \sin(\pi\, \widetilde{\sigma}_x) \cos(\pi\, \widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu}). \end{cases} \tag{6.9}$$

with the source term $\widetilde{s} = \widetilde{s}(\widetilde{x}, \widetilde{y})$ properly chosen so that the exact solution to the problem is given by

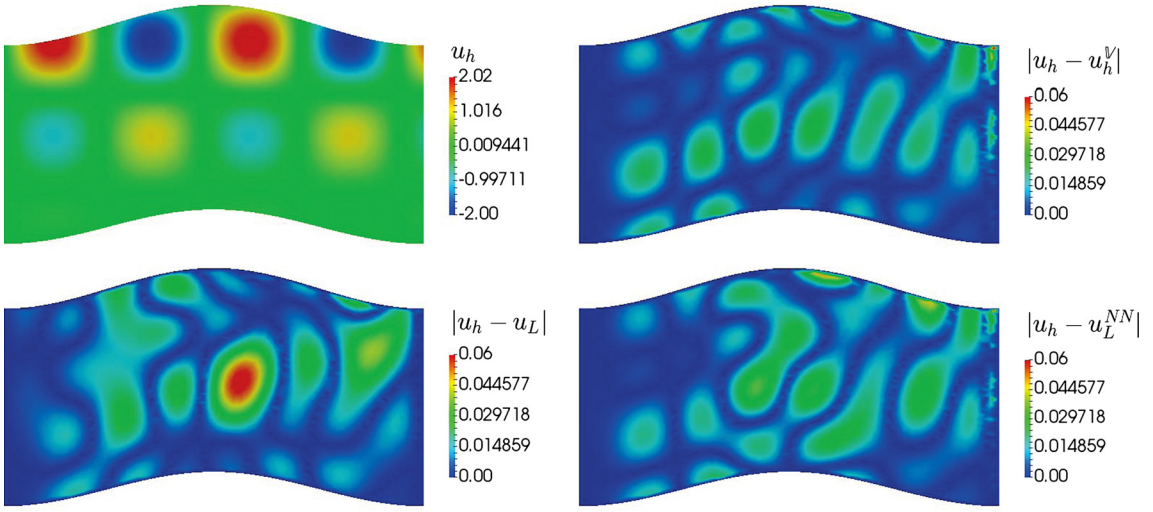


Fig. 6. FE solution (*top left*) to the Poisson problem (6.9) with $\boldsymbol{\mu} = (0.349, -0.413, 4.257)$, and pointwise errors yielded by either its projection onto V_{rb} (*top right*), or the POD-G method (*bottom left*), or the POD-NN method (*bottom right*). The results have been obtained by employing $L = 30$ POD modes.

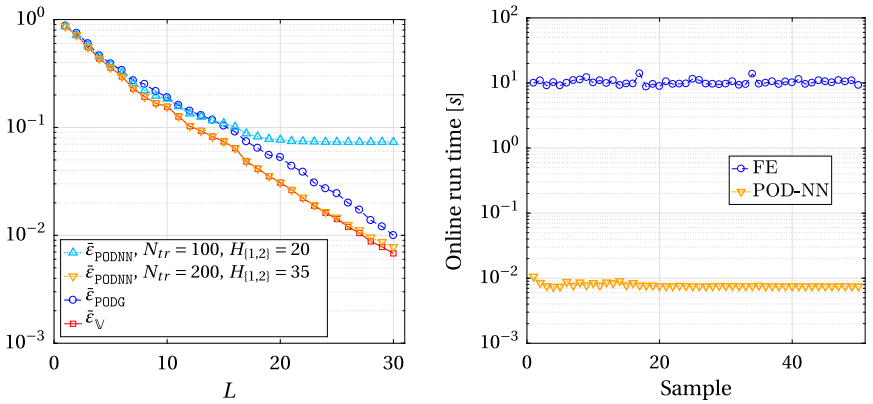


Fig. 7. Error analysis (*left*) and online CPU time (*right*) for the POD-G and the POD-NN methods applied to problem (6.9). The reduced basis have been generated via POD, relying on $N = 100$ snapshots. The second plot refers to RB models including $L = 30$ modal functions; within the POD-NN framework, an MLP embedding 35 neurons per inner layer has been used.

$$\widetilde{u}_{ex}(\widetilde{x}, \widetilde{y}) = \widetilde{y}\,\sin(\pi\,\widetilde{x})\cos(\pi\,\widetilde{y})$$

for all $\boldsymbol{\mu} \in \mathcal{P}$. Although the state equation has an exponential nonlinearity and the computational domain presents curvilinear boundaries, both of the RB methodologies studied in this work provide accurate solutions, close to the optimal one, i.e., $u_h^{\mathbb{V}}(\boldsymbol{\mu})$. This can be seen in Fig. 6, offering the finite element solution (*top left*) to (6.9) when $\boldsymbol{\mu} = (0.349, -0.413, 4.257)$, and the pointwise errors committed by either the projection onto V_{rb} (*top right*), or the POD-G method (*bottom left*), or the POD-NN method (*bottom right*). The computational mesh consists of 2792 nodes, resulting in $N_h = 2632$ degrees of freedom (as many as the inner nodes) for the FE scheme. The reduced space V_{rb} is generated by $L = 30$ basis functions, given by POD of $N = 100$ snapshots. Specifically, the results concerning the POD-NN method have been obtained by employing a neural network equipped with $H_1 = H_2 = 35$ neurons per hidden layer and trained with $N_{tr} = 200$ learning samples. In this way, the POD-NN procedure leads to an error which shows patterns similar to those featured by the projection error. This is not surprising, due to the way neural networks are trained within the POD-NN framework (see Section 5).

A quantitative evidence of the efficacy of the RB approximations is given in the plot on the left in Fig. 7, which reports the convergence analysis for both the POD-G and the POD-NN methods with respect to the number L of retained modal basis functions. As usual, the error has to be interpreted as an average over a parameter data set Ξ_{te}, here consisting of $N_{te} = 50$ samples. However, the second plot of Fig. 7 reveals how POD-NN produces a response for any online query approximately 10^3 times faster than POD-G does. On the other hand, the proposed methodology implies a larger offline computational cost. Here, while generating the POD basis takes less than half an hour, the MLPs learning phase (see Fig. 8, *left*) lasts around 8 hours, of which approximately one hour is devoted to compute the snapshots for training and validation.

Let us further analyze the first plot of Fig. 7. When the proper orthogonal decomposition of a set of $N = 100$ snapshots is coupled with a three-layers perceptron with 35 neurons per hidden layer, trained to approximate the map (5.2) relying

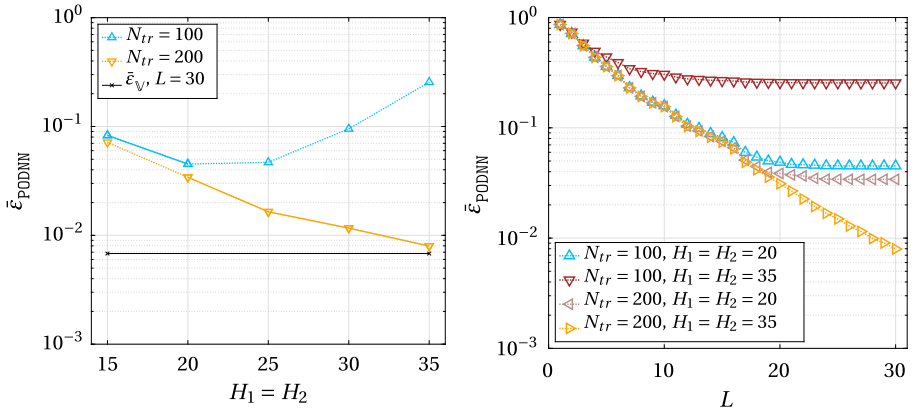


Fig. 8. Convergence analysis with respect to the number of hidden neurons (*left*) and modal functions (*right*) used within the POD-NN framework applied to problem (6.9). The results provided in the first plot have been obtained using $L = 30$ modes; the solid tracts refer to the steps performed by the automatic routine carried out to find an optimal network configuration.

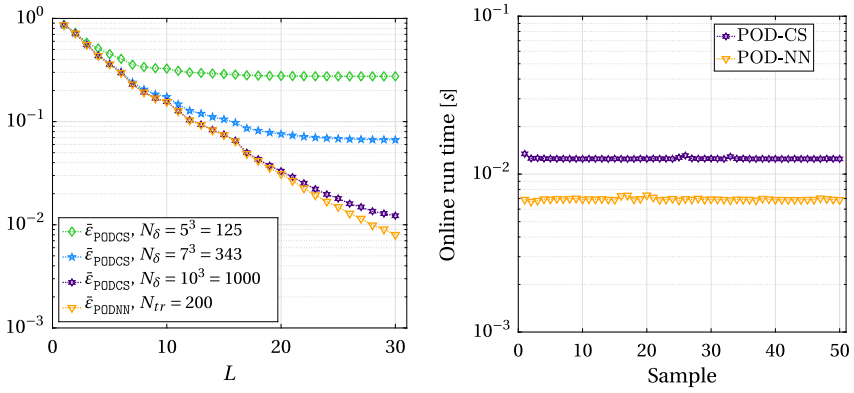


Fig. 9. Average relative errors (*left*) and online run times (*right*) on Ξ_{te} for the POD-CS and the POD-NN methods applied to problem (6.9). For the latter, the results refer to an MLP equipped with $H_1 = H_2 = 35$ neurons per hidden layer.

on $N_{tr} = 200$ learning data, the relative error is smaller than the error committed by the POD-Galerkin procedure for each $L \leq 30$. Yet, halfing the number of learning data employed, the POD-NN error curve stops decreasing at $L = 20$, then incurring a plateu. In this regard, Fig. 8 suggests that as the size of the available training set decreases, one should reduce the amount of computing units to be embodied in the network, to limit the risk of overfitting. Particularly, when only $N_{tr} = 100$ training samples are used, the optimal network configuration comprises 20 neurons per internal layer (*left*). However, doubling the dimension of the training set, the same configuration does not allow to fully exploit the augmented amount of information. In that case, a network with, e.g., 35 neurons per hidden layer is preferable (*right*).

The distinguishing and novel feature of the POD-NN method is represented by the employment of multi-layer perceptrons to recover the coefficients of the reduced model. To motivate this choice, let us pursue a more traditional approach in the interpolation step by resorting to cubic splines [14]. To this end, let $\Xi_\delta \subset \mathcal{P}$ be a tensor-product grid on the parameter domain, based on, e.g., Chebyshev nodes. In the offline phase, for each $\boldsymbol{\mu}_\delta \in \Xi_\delta$, we compute the truth solution $\mathbf{u}_h(\boldsymbol{\mu}_\delta) \in \mathbb{R}^{N_h}$ and we extract the expansion coefficients $\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}_\delta) \in \mathbb{R}^L$. At the online stage, given a new parameter value $\boldsymbol{\mu} \in \mathcal{P}$, the i-th expansion coefficient, $i = 1, \ldots, L$, is sought by cubic spline interpolation of the samples

$$\left\{\left(\boldsymbol{\mu}_\delta,\, \left(\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}_\delta)\right)_i\right)\right\}_{\boldsymbol{\mu}_\delta \in \Xi_\delta}.$$

Hence, denoting by $\mathbf{u}_{\text{rb}}^{\text{CS}}(\boldsymbol{\mu}) \in \mathbb{R}^L$ the approximation of $\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})$, the reduced order approximation of $\mathbf{u}_h(\boldsymbol{\mu})$ is given by $\mathbf{u}_L^{\text{CS}}(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\text{rb}}^{\text{CS}}(\boldsymbol{\mu})$. Similarly to the POD-G and the POD-NN methods, the accuracy of the resulting POD-CS procedure can be assessed by evaluating and averaging the relative error $\varepsilon_{\text{PODCS}}(L, \boldsymbol{\mu})$, defined as

$$\varepsilon_{\text{PODCS}}(L, \boldsymbol{\mu}) = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_L^{\text{CS}}(\boldsymbol{\mu})\right\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|} = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\,\mathbf{u}_{\text{rb}}^{\text{CS}}(\boldsymbol{\mu})\right\|}{\|\mathbf{u}_h(\boldsymbol{\mu})\|},$$

on a test parameter set $\Xi_{te} \subset \mathcal{P}$, with $\Xi_{te} \cap \Xi_\delta = \emptyset$.

For the Poisson problem (6.9), we report in the left plot of Fig. 9 the relative errors by both the POD-CS and the POD-NN methods. For the former, we provide the results obtained on tensor-product grids consisting of N_δ interpolation points,

with $N_\delta \in \{5^3, 7^3, 10^3\}$. For the latter, $H_1 = H_2 = 35$ neurons per hidden layer and $N_{tr} = 200$ training data have been used. Although the online performance of the two procedures are basically the same, as shown by the second plot in Fig. 9, we observe that the level of predictive accuracy enabled by the POD-NN method can be attained or at least approached by cubic spline interpolation only when this relies on $N_\delta = 1000$ samples. In fact, as mentioned in the Introduction, a standard interpolation technique may require a large number of samples to be able to enforce the constraints characterizing the nonlinear manifolds which the reduced bases typically belong to [1]. Hence, although this approach provides a valuable alternative for parametrized problems with a few parameters, it may be infeasible in real-life applications involving a large number of parameters. Conversely, the hope behind the choice of a neural network-based regression is that the amount of required snapshots properly scales with the dimension of the parameter space. This would justify the overheads due to the training phase.

6.2. Steady incompressible Navier–Stokes equations

The system of the Navier–Stokes equations model the conservation of mass and momentum for an incompressible Newtonian viscous fluid confined in a two- or three-dimensional region [45]. Letting $\boldsymbol{\mu} \equiv \boldsymbol{\mu}_g$ and $\widetilde{\boldsymbol{v}} = \widetilde{\boldsymbol{v}}(\widetilde{\boldsymbol{x}}; \boldsymbol{\mu})$ and $\widetilde{p} = \widetilde{p}(\widetilde{\boldsymbol{x}}; \boldsymbol{\mu})$ be, respectively, the velocity and pressure of the fluid, the parametrized steady version of the Navier–Stokes equations considered here reads

$$\begin{cases} \widetilde{\nabla} \cdot \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = 0 & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \quad \text{(a)} \\ -\nu(\boldsymbol{\mu})\, \widetilde{\Delta} \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + (\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \widetilde{\nabla}) \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \dfrac{1}{\rho(\boldsymbol{\mu})} \widetilde{\nabla} \widetilde{p}(\boldsymbol{\mu}) = \boldsymbol{0} & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \quad \text{(b)} \\ \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = \widetilde{\boldsymbol{h}} & \text{on } \widetilde{\Gamma}_D(\boldsymbol{\mu}), \quad \text{(c)} \\ \widetilde{p}(\boldsymbol{\mu}) \widetilde{\boldsymbol{n}} - \nu(\boldsymbol{\mu}) \widetilde{\nabla} \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \widetilde{\boldsymbol{n}} = \boldsymbol{0} & \text{on } \widetilde{\Gamma}_N(\boldsymbol{\mu}). \quad \text{(d)} \end{cases} \tag{6.10}$$

Here, $\widetilde{\boldsymbol{h}}$ denotes the velocity field prescribed on $\widetilde{\Gamma}_D(\boldsymbol{\mu})$, whereas homogeneous Neumann conditions are applied on $\widetilde{\Gamma}_N(\boldsymbol{\mu})$. Furthermore, $\rho(\boldsymbol{\mu})$ and $\nu(\boldsymbol{\mu})$ represent the uniform density and kinematic viscosity of the fluid, respectively. Note that, despite that these quantities encode physical properties, we let them depend on the geometrical parameters as well. Indeed, fluid dynamics can be characterized (and controlled) by means of a wealth of dimensionless quantities, e.g., the Reynold's number, which combine physical properties of the fluid with geometrical features of the domain. Therefore, a numerical study of the sensitivity of the system (6.10) with respect to $\boldsymbol{\mu}$ may be carried out by adapting either $\rho(\boldsymbol{\mu})$ or $\nu(\boldsymbol{\mu})$ as $\boldsymbol{\mu}$ varies, so to preserve a dimensionless quantity of interest.

To write the differential system (6.10) in weak form over a $\boldsymbol{\mu}$-independent configuration Ω, let us introduce the velocity and pressure spaces

$$\widetilde{X}(\boldsymbol{\mu}) = \left[H^1_{\widetilde{\Gamma}_D}\left(\widetilde{\Omega}(\boldsymbol{\mu})\right)\right]^d \quad \text{and} \quad \widetilde{Q}(\boldsymbol{\mu}) = L^2\left(\widetilde{\Omega}(\boldsymbol{\mu})\right),$$

respectively, with

$$X = \left[H^1_{\Gamma_D}(\Omega)\right]^d \quad \text{and} \quad Q = L^2(\Omega)$$

be their respective counterparts over Ω. By multiplying (6.10) by test functions $(\widetilde{\boldsymbol{\chi}}, \widetilde{\xi}) \in \widetilde{V}(\boldsymbol{\mu}) = \widetilde{X}(\boldsymbol{\mu}) \times \widetilde{Q}(\boldsymbol{\mu})$, integrating by parts and then tracing everything back onto Ω by means of the parametrized map $\boldsymbol{\Phi}(\cdot; \boldsymbol{\mu})$, we end up with the following parametrized weak variational problem: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u(\boldsymbol{\mu}) = (\boldsymbol{v}(\boldsymbol{\mu}), p(\boldsymbol{\mu})) \in V = X \times Q$ so that

$$a(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + d(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + b(p(\boldsymbol{\mu}), \nabla \cdot \boldsymbol{\chi}; \boldsymbol{\mu}) = f_1(\boldsymbol{\chi}; \boldsymbol{\mu}),$$
$$b(\nabla \cdot \boldsymbol{v}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) = f_2(\xi; \boldsymbol{\mu}),$$

for all $(\boldsymbol{\chi}, \xi) \in V$, with, for any $\boldsymbol{\mu} \in \mathcal{P}$,

$$c(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = \int_\Omega \left(\boldsymbol{\psi} \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla\right) \boldsymbol{\chi} \cdot \boldsymbol{\eta}\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega,$$
$$d(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = c(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{\psi}, \boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}),$$
$$a(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \nu(\boldsymbol{\mu}) \int_\Omega \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla \boldsymbol{\psi} \,:\, \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla \boldsymbol{\chi}\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega,$$
$$f_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = -a(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) - c(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}),$$
$$b(\boldsymbol{\psi}, \xi; \boldsymbol{\mu}) = -\frac{1}{\rho(\boldsymbol{\mu})} \int_\Omega \left(\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla \cdot \boldsymbol{\psi}\right) \xi\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega,$$
$$f_2(\xi; \boldsymbol{\mu}) = -b(\boldsymbol{l}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}),$$

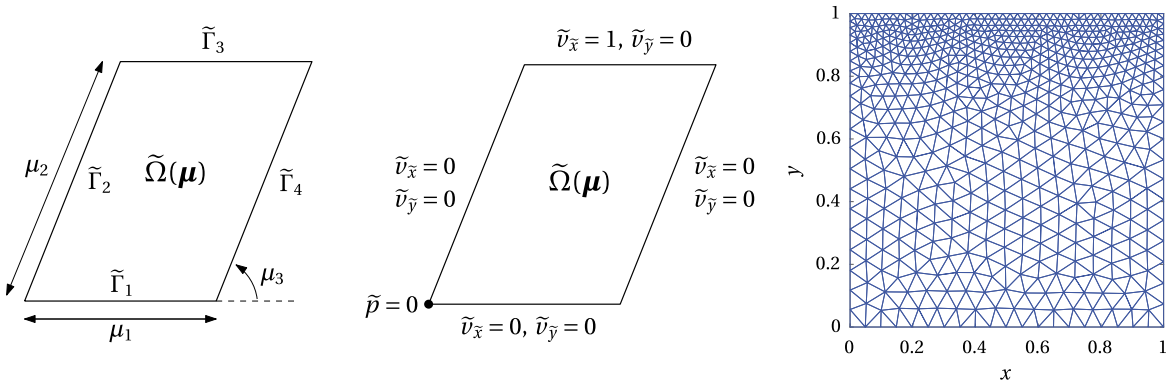


Fig. 10. Computational domain $\widetilde{\Omega}(\boldsymbol{\mu})$ (*left*), enforced velocity at the boundaries (*center*) and computational mesh Ω_h (*right*) for the lid-driven cavity problem for the incompressible Navier–Stokes equations.

for all $\boldsymbol{\psi},\ \boldsymbol{\chi},\ \boldsymbol{\eta} \in X$ and $\xi \in Q$. In the definitions of $d(\cdot, \cdot; \boldsymbol{\mu})$, $f_1(\cdot; \boldsymbol{\mu})$ and $f_2(\cdot; \boldsymbol{\mu})$, $\boldsymbol{l}(\boldsymbol{\mu}) \in \left[H^1(\Omega)\right]^d$ denotes the lifting vector field, with $\boldsymbol{l}(\boldsymbol{\mu})\big|_{\Gamma_D} = \boldsymbol{h}(\boldsymbol{\mu})$, $\boldsymbol{h}(\boldsymbol{x}; \boldsymbol{\mu}) = \widetilde{\boldsymbol{h}}(\boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\mu}))$ being the velocity field prescribed on Γ_D. Hence, the weak solution to (6.10) defined over the fixed domain Ω is given by $\left(\boldsymbol{v}(\boldsymbol{\mu}) + \boldsymbol{l}(\boldsymbol{\mu}), p(\boldsymbol{\mu})\right)$.

6.2.1. The lid-driven cavity problem

In this subsection, we are concerned with the numerical simulation of a viscous flow within a parallelogram-shaped cavity, illustrated in Fig. 10 (*left*). The geometry of the domain is affected by three parameters: $\mu_1 \in [1, 2]$ and $\mu_2 \in [1, 2]$ define the length of the horizontal and slanting (possibly vertical) edges, respectively, whereas $\mu_3 \in \left[\pi/6, 5\pi/6\right]$ denotes the angle between the oblique sides and the positive $\tilde{x}$-semiaxis. The Dirichlet boundary conditions enforced on the velocity field, graphically represented in Fig. 10 (*center*), are such that the flow is driven by a unit horizontal velocity at the top boundary [41]. Hence, this benchmark is typically referred to as the *lid-driven cavity* problem. In addition, we fix the pressure at the lower-left corner, in order to retain the uniqueness of the solution [16]. Therefore, the computational mesh Ω_h (Fig. 10, *right*) is refined in the upper part of the domain, so to properly resolve the steep velocity gradient near the upper boundary and the pressure singularities at the top corners.

For the Navier–Stokes equations, a suitable choice of the FE spaces is crucial to fulfill the well-known *inf-sup* stability condition [45]. A common and effective choice consists in using quadratic finite elements for the components of the velocity field and linear finite elements for the pressure, leading to the so-called $\mathbb{P}^2 - \mathbb{P}^1$ (or Taylor–Hood) FE discretization [41]. Fig. 11 shows the velocity streamlines (*top*) and the pressure distribution (*bottom*) obtained through the resulting FE method for, from the left to the right, $\boldsymbol{\mu} = \left(1,\ 2/\sqrt{3},\ 2\pi/3\right)$, $\boldsymbol{\mu} = \left(1,\ 1,\ \pi/2\right)$ or $\boldsymbol{\mu} = \left(1,\ 2/\sqrt{3},\ \pi/6\right)$. Observe how the moving lid induces a large velocity gradient close to the top boundary, while it only slightly affects the fluid in the lower part of the cavity. In turn, this gives rise to two vortices, whose extent highly depends on the shape of the domain, or, equivalently, $\boldsymbol{\mu}$. Conversely, the pressure field does not undergo noticeable alterations across the parameter space. For all the configurations, a low-pressure region takes place at the upper-left corner of the domain and in the center of the major vortex (although it may be not clearly visible). Instead, the upper-right corner represents a stagnation point for the $\tilde{x}$-velocity, and so therein the pressure assumes values larger than in the rest of the cavity [16]. All the results presented in Fig. 11 refer to a Reynold's number of 400. For the specific problem at hand, the Reynold's number reads:

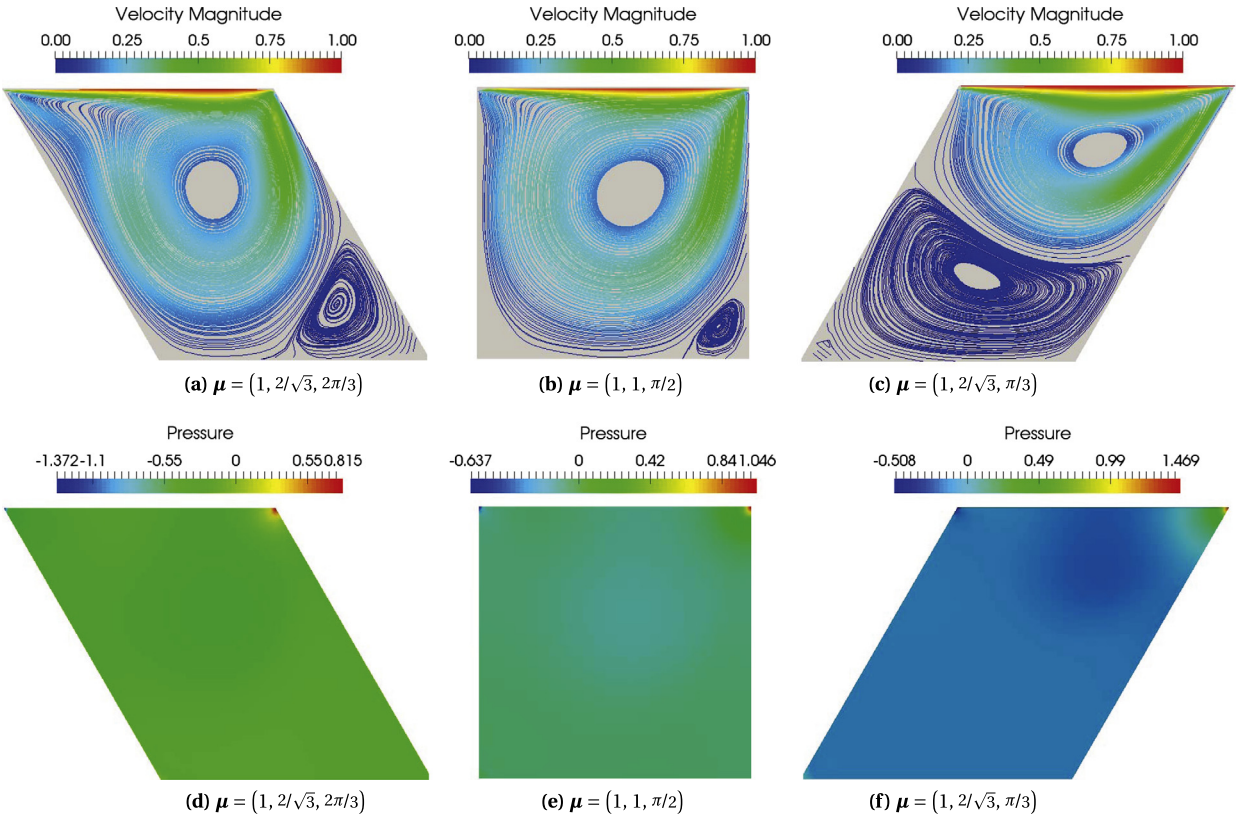
$$Re = \frac{\max\left\{\mu_1,\ \mu_2\right\}}{\nu(\boldsymbol{\mu})}.$$

In our analyses, $\nu(\boldsymbol{\mu})$ is adapted as the geometry varies so that the Reynold's number is either 200 or 400.

Regarding the reduced order modeling of the parametrized problem of interest, we slightly distance ourselves from the general workflow depicted in Sections 3 and 5. Indeed, for the Navier–Stokes equations it is convenient to construct two separate reduced spaces [2,11,44]: X_{rb}, of dimension $L_{\boldsymbol{v}}$, for the velocity field and Q_{rb}, of dimension L_p, for the pressure distribution, respectively represented by the matrices

$$\mathbb{V}_{\boldsymbol{v}} = \left[\boldsymbol{\psi}_1^{\boldsymbol{v}} \,\middle|\, \dots \,\middle|\, \boldsymbol{\psi}_{L_{\boldsymbol{v}}}^{\boldsymbol{v}}\right] \in \mathbb{R}^{N_h^{\boldsymbol{v}} \times L_{\boldsymbol{v}}} \quad \text{and} \quad \mathbb{V}_p = \left[\boldsymbol{\psi}_1^{p} \,\middle|\, \dots \,\middle|\, \boldsymbol{\psi}_{L_p}^{p}\right] \in \mathbb{R}^{N_h^{p} \times L_p},$$

with $N_h^{\boldsymbol{v}}$ (respectively, N_h^p) the dimension of the FE velocity (resp., pressure) space. Although the underpinning finite element solver has been designed to fulfill the inf-sup condition, thus ensuring the stability of the scheme, when dealing with incompressible flows this property may not be automatically inherited by the POD-Galerkin reduced order solver. As a result, it may show severe instabilities [8]. Indeed, one has to carefully choose the reduced velocity space X_{rb} so to meet an equivalent of the inf-sup condition at the reduced level. To this end, in our simulations we resort to a *supremizer*



(a) $\boldsymbol{\mu} = \left(1, 2/\sqrt{3}, 2\pi/3\right)$ **(b)** $\boldsymbol{\mu} = \left(1, 1, \pi/2\right)$ **(c)** $\boldsymbol{\mu} = \left(1, 2/\sqrt{3}, \pi/3\right)$

(d) $\boldsymbol{\mu} = \left(1, 2/\sqrt{3}, 2\pi/3\right)$ **(e)** $\boldsymbol{\mu} = \left(1, 1, \pi/2\right)$ **(f)** $\boldsymbol{\mu} = \left(1, 2/\sqrt{3}, \pi/3\right)$

Fig. 11. Velocity streamlines (*top*) and pressure distribution (*bottom*) for the lid-driven cavity problem, computed through the FE method. Three different parameter values are considered; for all configurations, the Reynold's number is 400.

enrichment of X_{rb}, as proposed in [2]. At each pressure snapshot $\mathbf{p}_h\left(\boldsymbol{\mu}^{(n)}\right) \in \mathbb{R}^{N_h^p}$, $n = 1, \ldots, N$, we associate the *supremizer solution* $\mathbf{s}_h\left(\boldsymbol{\mu}^{(n)}\right) \in \mathbb{R}^{N_h^{\boldsymbol{v}}}$, defined as the solution to a linear system of the form

$$\mathbb{A}\, \mathbf{s}_h\left(\boldsymbol{\mu}^{(n)}\right) = \mathbb{B}\left(\boldsymbol{\mu}^{(n)}\right) \mathbf{p}_h\left(\boldsymbol{\mu}^{(n)}\right),$$

with $\mathbb{A} \in \mathbb{R}^{N_h^{\boldsymbol{v}} \times N_h^{\boldsymbol{v}}}$ and $\mathbb{B}\left(\boldsymbol{\mu}^{(n)}\right) \in \mathbb{R}^{N_h^{\boldsymbol{v}} \times N_h^p}$. Letting

$$\mathbb{V}_{\boldsymbol{s}} = \left[\boldsymbol{\psi}_1^{\boldsymbol{s}} \,\middle|\, \ldots \,\middle|\, \boldsymbol{\psi}_{L_{\boldsymbol{s}}}^{\boldsymbol{s}}\right] \in \mathbb{R}^{N_h^{\boldsymbol{v}} \times L_{\boldsymbol{s}}}$$

be the POD basis of rank $L_{\boldsymbol{s}}$ extracted from the ensemble of snapshots $\left\{\mathbf{s}_h\left(\boldsymbol{\mu}^{(1)}\right), \ldots, \mathbf{s}_h\left(\boldsymbol{\mu}^{(N)}\right)\right\}$, the POD-G approximation of the velocity field is then sought in the form

$$\bar{\mathbf{v}}_L(\boldsymbol{\mu}) = \overline{\mathbb{V}}_{\boldsymbol{v}}\, \bar{\mathbf{v}}_{\mathrm{rb}}(\boldsymbol{\mu}) = \sum_{i=1}^{L_{\boldsymbol{v}}} v_{\mathrm{rb}}^{(i)} \boldsymbol{\psi}_i^{\boldsymbol{v}} + \sum_{i=1}^{L_{\boldsymbol{s}}} s_{\mathrm{rb}}^{(i)} \boldsymbol{\psi}_i^{\boldsymbol{s}},$$

where

$$\overline{\mathbb{V}}_{\boldsymbol{v}} = \left[\mathbb{V}_{\boldsymbol{v}}\ \mathbb{V}_{\boldsymbol{s}}\right] \in \mathbb{R}^{N_h^{\boldsymbol{v}} \times (L_{\boldsymbol{v}} + L_{\boldsymbol{s}})} \quad \text{and} \quad \bar{\mathbf{v}}_{\mathrm{rb}}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{v}_{\mathrm{rb}}(\boldsymbol{\mu}) \\ \mathbf{s}_{\mathrm{rb}}(\boldsymbol{\mu}) \end{bmatrix} \in \mathbb{R}^{L_{\boldsymbol{v}} + L_{\boldsymbol{s}}}.$$

In [2], several numerical evidences suggest that a proper value for $L_{\boldsymbol{s}}$ should lay in between $L_p/2$ and L_p. In our simulations, we set $L_{\boldsymbol{s}} = L_p = L_{\boldsymbol{v}}$, leading to a stable reduced (nonlinear) system in $L = 3\, L_{\boldsymbol{v}}$ unknowns.

Concerning the POD-NN procedure, for any $\boldsymbol{\mu} \in \mathcal{P}$, reduced order discretizations $\mathbf{v}_{L_{\boldsymbol{v}}}^{\mathrm{NN}}(\boldsymbol{\mu})$ and $\mathbf{p}_{L_p}^{\mathrm{NN}}(\boldsymbol{\mu})$ of $\boldsymbol{v}(\boldsymbol{\mu})$ and $p(\boldsymbol{\mu})$, respectively, are sought in the form

$$\mathbf{v}_{L_{\boldsymbol{v}}}^{\mathrm{NN}}(\boldsymbol{\mu}) = \mathbb{V}_{\boldsymbol{v}}\, \mathbf{v}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu}) \quad \text{and} \quad \mathbf{p}_{L_p}^{\mathrm{NN}}(\boldsymbol{\mu}) = \mathbb{V}_p\, \mathbf{p}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu}).$$

Here, $\mathbf{v}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu})$ and $\mathbf{p}_{\mathrm{rb}}^{\mathrm{NN}}(\boldsymbol{\mu})$ denote the output vectors provided by two *distinct* multi-layer perceptrons, respectively trained with the ensemble of input-output patterns

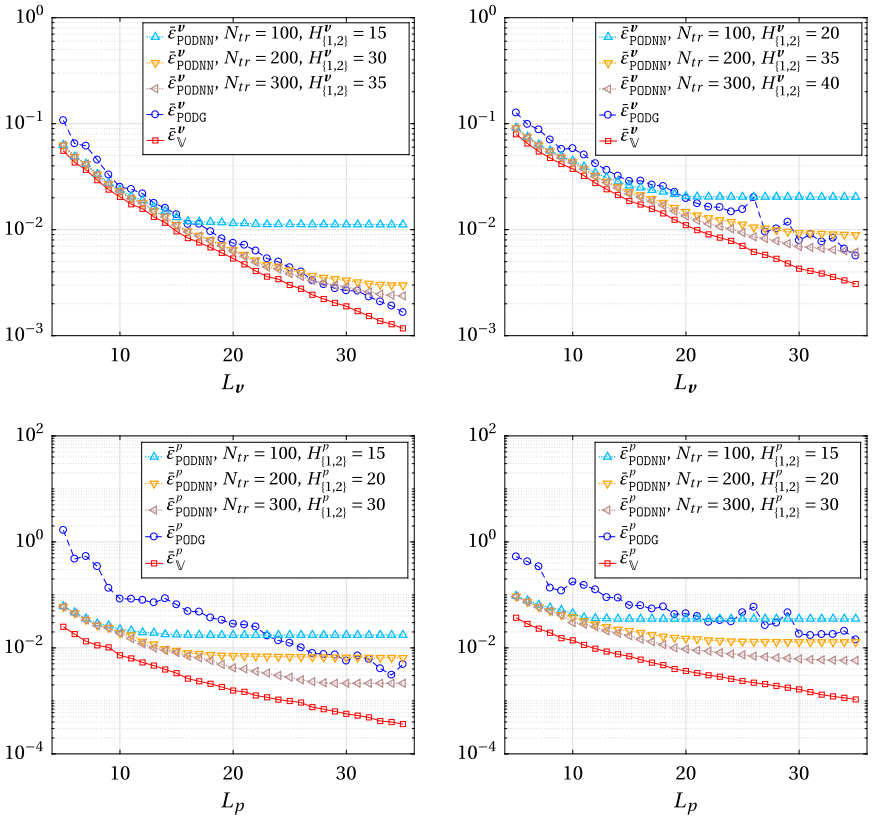


Fig. 12. Velocity (*top*) and pressure (*bottom*) error analysis for the POD-G and POD-NN methods applied to the lid-driven cavity problem with $Re = 200$ (*left*) and $Re = 400$ (*right*).

$$\left\{\boldsymbol{\mu}^{(i)},\, \mathbb{V}_{\boldsymbol{v}}^T \mathbf{v}_h(\boldsymbol{\mu}^{(i)})\right\}_{1\le i\le N_{tr}} \quad \text{and} \quad \left\{\boldsymbol{\mu}^{(i)},\, \mathbb{V}_p^T \mathbf{p}_h(\boldsymbol{\mu}^{(i)})\right\}_{1\le i\le N_{tr}}.$$

At this regard, let us point out two important remarks.

(i) The supremizer solutions are not involved (either directly or indirectly) in the POD-NN framework. Indeed, they ensure the overall stability of the POD-G procedure, but they do not improve the accuracy of the method, and so they can be disregarded whenever stability is not an issue.

(ii) The routine outlined in Section 5, aiming at finding an optimal network configuration, is applied *separately* to the perceptrons designated to predict the velocity field and to the perceptrons required to approximate the pressure distribution. As a result, once the offline phase of the POD-NN method is completed, we end up with two different networks, equipped with $H_1^{\boldsymbol{v}} = H_2^{\boldsymbol{v}}$ and $H_1^p = H_2^p$ computing units per hidden layer, respectively.

Fig. 12 reports the error committed by both RB procedures when approximating the velocity field (*top*) and the pressure distribution (*bottom*) by means of $L_{\boldsymbol{v}}$ velocity modes, L_p pressure modes and, exclusively for the POD-Galerkin method, $L_{\boldsymbol{s}}$ supremizer modes, with $5 \le L_{\boldsymbol{v}} = L_{\boldsymbol{s}} = L_p \le 35$. The plots on the left refer to a Reynold's number of 200, the ones on the right to a Reynold's number of 400. Note that for clarity of illustration, the symbols denoting the projection, POD-G and POD-NN errors have been identified with a superscript (either $\boldsymbol{v}$ or p) recalling the state variable they refer to.

While the error yielded on the velocity field shows an almost perfect exponential decay with the number of POD modes included in the RB model, the POD-G method faces difficulties in providing a correct recovery of the pressure, already for $Re = 200$. Indeed, the corresponding error curve is not monotone, and generally is at least one order of magnitude above the projection error. Conversely, the POD-NN method attains a satisfactory predictive accuracy. The advantages of resorting to a neural network-based nonlinear regression coupled with POD are evident when the approximation is built upon a few basis functions, say $L_p < 20$. Moreover, the proposed routine recommends the use of fewer neurons to predict the POD coefficients for the pressure than for the velocity, thus resulting in a lighter perceptron.

This is confirmed also by Fig. 13, which provides a sensitivity analysis of the predictive accuracy featured by the POD-NN method with respect to the amount of neurons and training samples used. Observe that for the pressure (*bottom*), employing more than 30 neurons within each hidden layer is counter-productive, both for $Re = 200$ (*left*) and $Re = 400$ (*right*). This assertion agrees with the peculiar role played by the pressure in the parametrized lid-driven cavity problem, with similar patterns featured across the entire parameter domain.

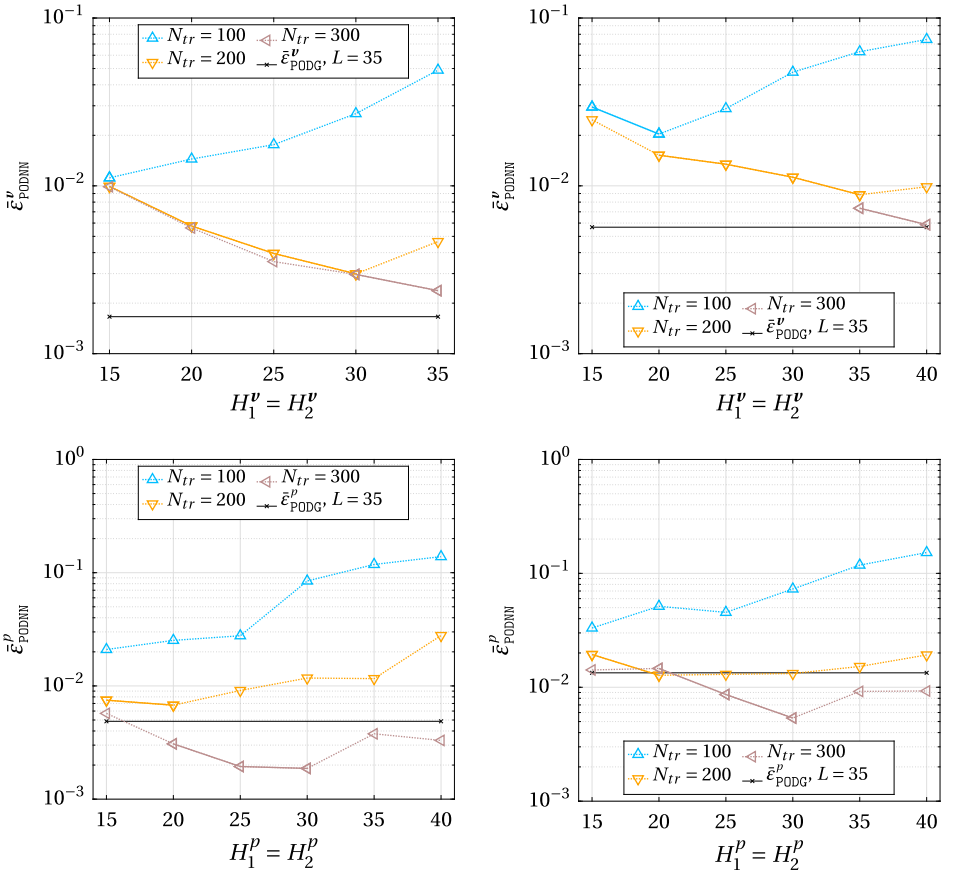


Fig. 13. Convergence analysis with respect to the number of hidden neurons and training samples used within the POD-NN procedure to approximate the velocity field (*top*) and the pressure distribution (*bottom*) by means of 35 modal functions. The Reynold's number is either 200 (*left*) or 400 (*right*).

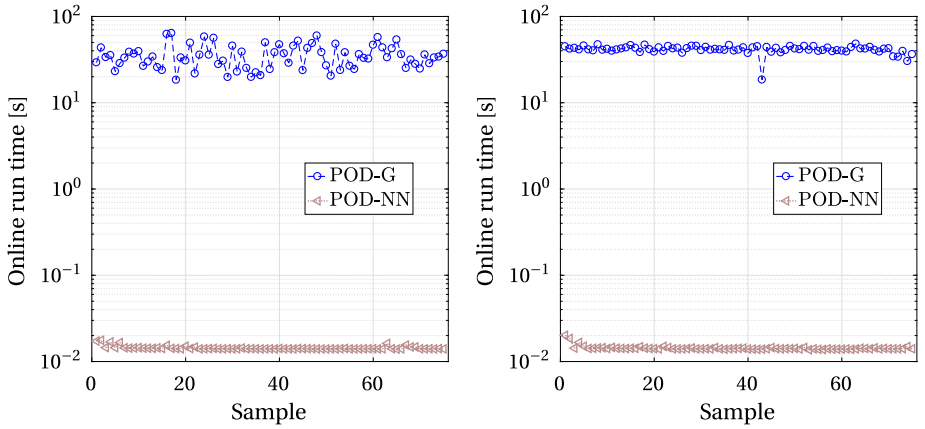


Fig. 14. Online run times for the POD-G and the POD-NN method applied to the lid-driven cavity problem with $Re = 200$ (*left*) and $Re = 400$ (*right*). $N_{te} = 75$ test configurations are considered. For the POD-NN method, the reported times include the (sequential) evaluation of both neural networks for the velocity and pressure field.

On the contrary, we have seen that the velocity field presents more complex dynamics, highly varying with the domain configuration. As a result, an optimal approximation of the velocity is obtained for the maximum values of $H^{\boldsymbol{v}}_i$, $i = 1, 2$, and N_{tr} tested, that is, $H^{\boldsymbol{v}}_1 = H^{\boldsymbol{v}}_2 = 35$ and $N_{tr} = 300$ for $Re = 200$, $H^{\boldsymbol{v}}_1 = H^{\boldsymbol{v}}_2 = 40$ and $N_{tr} = 300$ for $Re = 400$. Moreover, we may not be able to exactly attain the same precision enabled by the standard POD-Galerkin procedure, although the results are quite similar. However, this (slight) loss of accuracy is offset by a (great) reduction in the online run time: the

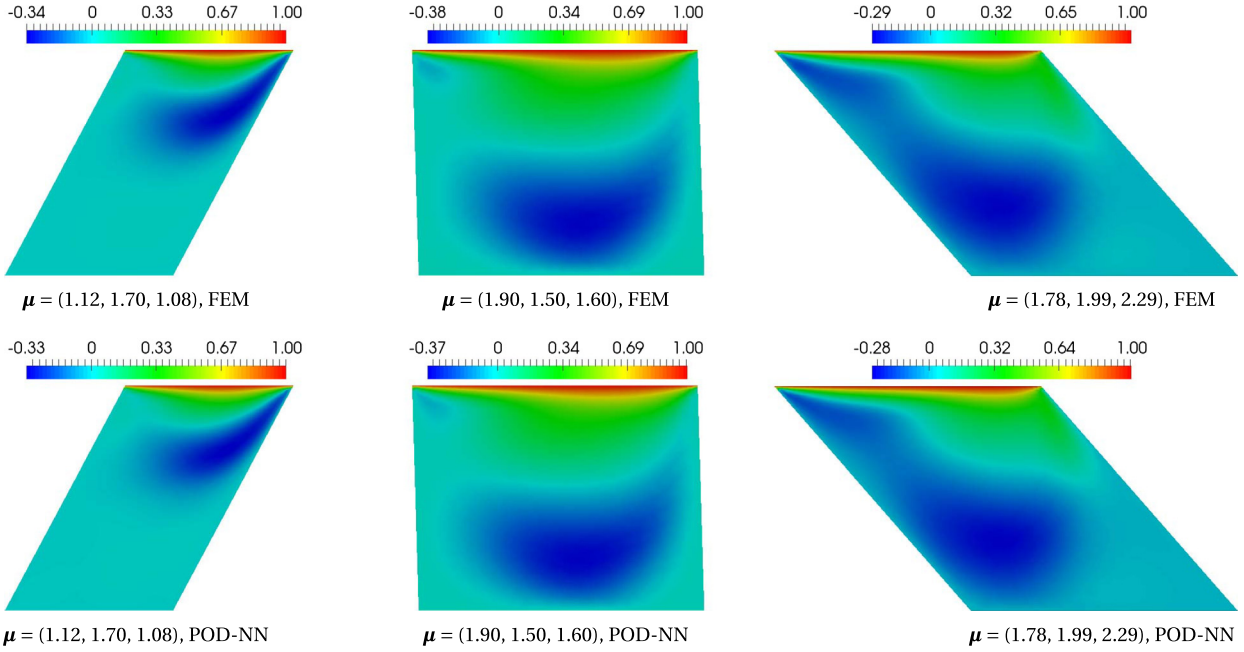


Fig. 15. $\widetilde{x}$-velocity contour at three parameter values, as computed through the FE (*top*) and POD-NN (*bottom*) method. For each configuration, the Reynold's number is 400.

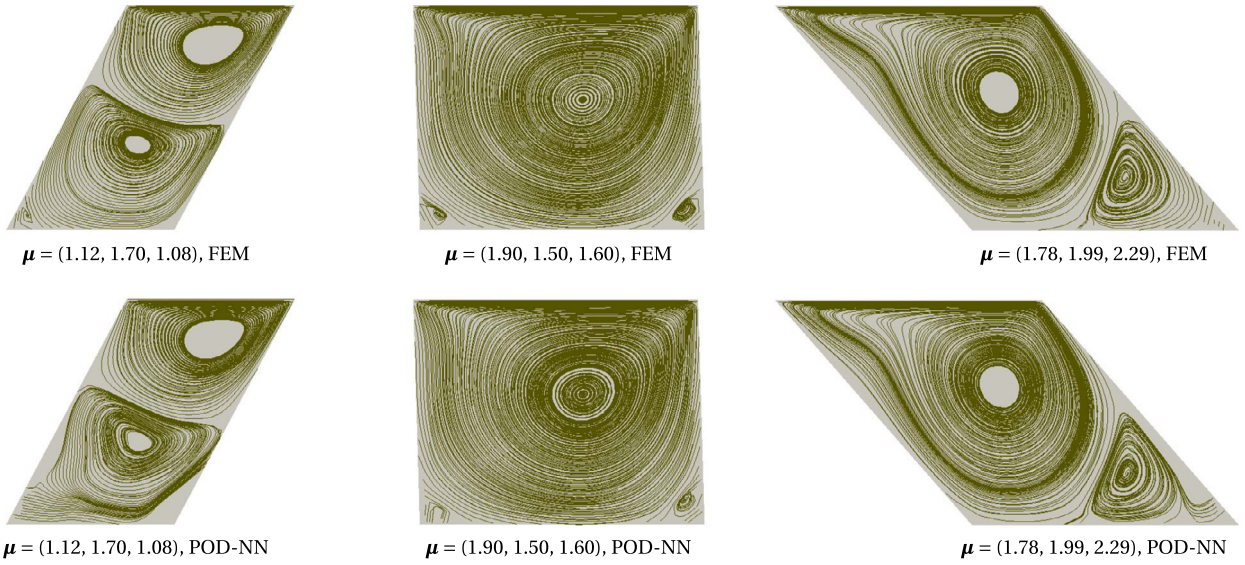


Fig. 16. Streamlines at three parameter values, as computed through the FE (*top*) and POD-NN (*bottom*) method. For each configuration, the Reynold's number is 400.

POD-NN method takes around 2/100 of seconds per query, against the average 40 seconds required by the POD-G method; see Fig. 14. As for the test cases for the Poisson problem, this comes at the cost of a longer offline phase.

Another numerical evidence of the predictive accuracy of the proposed POD-NN RB method is provided in Fig. 15, showing the contour plots for the $\widetilde{x}$-velocity computed through the FE (*top*) and the POD-NN (*bottom*) scheme. Three different configurations, corresponding to as many input vectors, are considered; the Reynold's number is fixed to 400. We can appreciate good agreement between the solutions given by the full-order and the reduced order methods.

Lastly, Fig. 16 compares the streamlines obtained through the direct method (*top*) and the proposed reduced basis approach (*bottom*) over the three configurations previously considered. Streamlines provide an interesting test, as minor variations in velocity contours may lead to substantial differences in the streamlines [11]. Nevertheless, we observe a good agreement between the two methods. In particular, in the second example, the POD-NN method is still able to detect the two micro recirculation zones at the lower corners of the domain. On the other hand, the method partially fails in properly

describing the velocity field at the bottom-left corner in the first configuration and at the bottom-right corner in the last configuration. However, these are effectively dead zones, and one can safely disregard these little imprecisions.

7. Conclusion

In this work, we propose a non-intrusive RB method (referred to as POD-NN) for parametrized steady-state PDEs. The method extracts a reduced basis from a collection of snapshots through a POD procedure and employs multi-layer perceptrons to approximate the coefficients of the reduced model. By exploiting the fundamental results by Cybenko (see Subsection 4.2), we limit ourselves to perceptrons endowed with two hidden layers. The identification of the optimal number of inner neurons and the minimum amount of training samples to prevent overfitting is performed during the offline stage through an automatic routine, relying upon the Latin hypercube sampling and the Levenberg–Marquardt training algorithm. On the one hand, this guarantees a complete decoupling between the offline and the online phase, with the latter having a computational cost independent of the dimension of the full-order model. On the other hand, this extends the offline stage with respect to a standard projection-based RB procedure, making the POD-NN method practically convenient only when the underlying PDE has to be solved for many parameter values (many-query context).

The POD-NN method has been successfully tested on the nonlinear Poisson equation in one and two spatial dimensions, and on two-dimensional cavity viscous flows, modeled through the steady incompressible Navier–Stokes equations. In particular, the proposed RB strategy enables the same predictive accuracy provided by the POD-Galerkin method while reducing the CPU time required to process an online query by two or even three order of magnitudes.

All test cases considered in our numerical studies involved three parameters, affecting either physical or geometrical factors of the differential problem. The extension of the POD-NN method to time-dependent problems depending on many parameters is left as future work.

Lastly, let us point out that although in this work we used POD to recover a reduced space, this choice is not binding, and a greedy approach may also be pursued.

References

[1] D. Amsallem, Interpolation on Manifolds of CFD-Based Fluid and Finite Element-Based Structural Reduced-Order Models for On-line Aeroelastic Predictions, Doctoral dissertation, Department of Aeronautics and Astronautics, Stanford University, 2010.
[2] F. Ballarin, A. Manzoni, A. Quarteroni, G. Rozza, Supremizer Stabilization of POD-Galerkin Approximation of Parametrized Navier–Stokes Equations, MATHICSE technical report, École Polytechnique Fédérale de Lausanne, 2014.
[3] M. Barrault, Y. Maday, N.C. Nguyen, A.T. Patera, An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations, C. R. Math. 339 (9) (2004) 667–672.
[4] V. Barthelmann, E. Novak, K. Ritter, High-dimensional polynomial interpolation on sparse grids, Adv. Comput. Math. 12 (4) (2000) 273–288.
[5] M.P. Bendsøe, O. Sigmund, Topology Optimization: Theory, Methods and Applications, Springer Science & Business Media, Heidelberg, 2004.
[6] P.F. Brown, V.J.D. Pietra, S.A.D. Pietra, R.L. Mercer, The mathematics of statistical machine translation: parameter estimation, Comput. Linguist. 19 (2) (1993) 263–311.
[7] A. Buffa, Y. Maday, A.T. Patera, C. Prud'Homme, G. Turinici, A priori convergence of the greedy algorithm for the parametrized reduced basis method, ESAIM: Math. Model. Numer. Anal. 46 (2012) 595–603.
[8] J. Burkardt, M. Gunzburger, H.C. Lee, POD and CVT-based reduced-order modeling of Navier–Stokes flows, Comput. Methods Appl. Mech. Eng. 196 (2006) 337–355.
[9] F. Casenave, A. Ern, T. Lelièvre, A nonintrusive reduced basis method applied to aeroacoustic simulations, Adv. Comput. Math. 41 (2015) 961–986.
[10] S. Chaturantabut, D.C. Sorensen, Nonlinear model reduction via discrete empirical interpolation, SIAM J. Sci. Comput. 32 (5) (2010) 2737–2764.
[11] W. Chen, J.S. Hesthaven, B. Junqiang, Z. Yang, Y. Tihao, A greedy non-intrusive reduced order model for fluid dynamics, J. Northwest. Polytech. Univ. (2017).
[12] G. Cybenko, Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient, Technical report, Department of Computer Science, Tufts University, 1988.
[13] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314.
[14] C. De Boor, A Practical Guide to Splines, Springer-Verlag, New York, NY, 1978.
[15] S. Deparis, Reduced basis error bound computation of parameter-dependent Navier–Stokes equations by the natural norm approach, SIAM J. Numer. Anal. 46 (4) (2008) 2039–2067.
[16] G. Dhondt, CalculiX CrunchiX user's manual, available at http://web.mit.edu/calculix_v2.7/CalculiX/ccx_2.7/doc/ccx/node1.html, 2014.
[17] C. Eckart, G. Young, The approximation of one matrix by another of lower rank, Psychometrika 1 (1936) 211–218.
[18] J.L. Eftang, Reduced Basis Methods for Partial Differential Equations, Master thesis, Department of Mathematical Sciences, Norwegian University of Science and Technology, 2008.
[19] P.E. Gill, W. Murray, M.H. Wright, Practical Optimization, Academic Press, 1981.
[20] M.T. Hagan, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Netw. 5 (6) (1994) 989–993.
[21] M.T. Hagan, H.B. Demuth, M.H. Beale, Neural Network Design, PWS Publishing, Boston, MA, 1996.
[22] S. Haykin, Neural Networks: A comprehensive Foundation, Prentice Hall, Upper Saddle River, NJ, 2004.
[23] J.S. Hesthaven, B. Stamn, G. Rozza, Certified Reduced Basis Methods for Parametrized Partial Differential Equations, Springer, New York, NY, 2016.
[24] J.S. Hesthaven, B. Stamn, S. Zhang, Efficienty greedy algorithms for high-dimensional parameter spaces with applications to empirical interpolation and reduced basis methods, ESAIM: Math. Model. Numer. Anal. 48 (1) (2014) 259–283.
[25] R.L. Imam, Latin hypercube sampling, in: Encyclopedia of Quantitative Risk Analysis and Assessment, 2008.
[26] C. Jaggli, L. Iapichino, G. Rozza, An improvement on geometrical parametrizations by transfinite maps, C. R. Acad. Sci. Paris, Ser. I 352 (2014) 263–268.
[27] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the 40th International Joint Conference on Artificial Intelligence, vol. 2, 1995, pp. 1137–1143.
[28] D. Kriesel, A brief introduction to neural networks, retrieved from http://www.dkriesel.com/en/science/neural_networks, 2007.
[29] O. Le Maître, O.M. Knio, Spectral Methods for Uncertainty Quantification with Applications to Computational Fluid Dynamics, Springer Science & Business Media, Berlin, 2010.

[30] K. Levenberg, A method for the solution of certain non-linear problems in least squares, Q. Appl. Math. 2 (1944) 164–168.
[31] E.B. Lee, L. Markus, Foundations of Optimal Control Theory, John Wiley & Sons, New York, NY, 1967.
[32] Y.C. Liang, H.P. Lee, S.P. Lim, W.Z. Lin, K.H. Lee, C.G. Wu, Proper orthogonal decomposition and its applications, part I: theory, J. Sound Vib. 252 (3) (2002) 527–544.
[33] Y. Maday, Reduced basis method for the rapid and reliable solution of partial differential equations, in: Proceedings of the International Congress of Mathematicians, Madrid, Spain, 2006, pp. 1255–1269.
[34] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, J. Soc. Ind. Appl. Math. 11 (2) (1963) 431–441.
[35] Inc The MathWorks, Machine learning challenges: choosing the best model and avoiding overfitting, retrieved from https://it.mathworks.com/campaigns/products/offer/common-machine-learning-challenges.html, 2016.
[36] W. Mitchell, M.A. McClain, A collection of 2D elliptic problems for testing adaptive algorithms, in: NISTIR, 2010, p. 7668.
[37] A. Manzoni, F. Negri, Automatic Reduction of PDEs Defined on Domains with Variable Shape, MATHICSE technical report, École Polytechnique Fédérale de Lausanne, 2016.
[38] M.D. Møller, A scaled conjugate gradient algorithm for fast supervised learning, Neural Netw. 6 (1993) 525–533.
[39] M.A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.
[40] F. Negri, A. Manzoni, D. Amsallem, Efficient model reduction of parametrized systems by matrix discrete empirical interpolation, J. Comput. Phys. 303 (2015) 431–454.
[41] P.O. Persson, Implementation of Finite-Element Based Navier–Stokes Solver, Massachussets Institue of Technology, 2002.
[42] C. Prud'homme, D.V. Rovas, K. Veroy, L. Machiels, Y. Maday, A.T. Patera, G. Turinici, Reliable real-time solution of parametrized partial differential equations: reduced-basis output bound methods, J. Fluids Eng. 124 (1) (2002) 70–80.
[43] A. Quarteroni, Numerical Models for Differential Problems, vol. 2, Springer Science & Business Media, New York, NY, 2010.
[44] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: An Introduction, Springer, New York, NY, 2015, 2015.
[45] R. Rannacher, Finite Element Methods for the Incompressible Navier–Stokes Equations, Lecture Notes, Institute of Applied Mathematics, University of Heidelberg, 1999.
[46] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (1958) 386–408.
[47] E. Schmidt, Zur theorie der linearen und nichtlinearen integralgleichungen. I. Teil: Entwicklung willkürlicher funktionen nach systemen vorgeschriebener, Math. Ann. 63 (1907) 433–476.
[48] C. Stergiou, D. Siganos, Neural networks, retrieved from https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introductiontoneuralnetworks, 2013.
[49] S. Volkwein, Model Reduction Using Proper Orthogonal Decomposition, Lecture Notes, Institute of Applied Mathematics, University of Konstanz, 2008.