A non-intrusive reduced basis method using artificial neural networks

J.S. Hesthaven, S. Ubbiali

École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

Abstract

In this work, we develop a non-intrusive reduced basis (RB) method for parametrized time-indepedent partial differential equations (PDEs). The proposed method extracts a reduced basis from a collection of high-fidelity solutions via 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 inner neurons and the minimum amount of training samples which avoids 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 training algorithm. This guarantees a complete offline-online decoupling, leading to an efficient RB method - referred to as POD-NN - suitable also for nonlinear problems featuring a non-affine parametric dependence. Numerical studies are presented for the linear and nonlinear Poisson equation and for driven cavity viscous flows, modeled through the steady uncompressible 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 with respect to a traditional RB strategy based on the Galerkin projection process.

Keywords: non-intrusive reduced basis method, proper orthogonal decomposition, multi-layer perceptron, Levenberg-Marquardt algorithm, Poisson equation, driven cavity flow

1. Introduction

Several applications arising in engineering and applied sciences involve mathematical models expressed as parametrized partial differential equations (PDEs), in which boundary conditions, material properties, source terms, loads or geometrical factors of the underlying physical problem are addressed to a parameter $\boldsymbol{\mu}$ [19, 30, 34]. A list of notable examples includes parameter estimation [6], topology optimization [5], optimal control [40] and uncertainty quantification [39]. In these contexts, one is typically interested in a real-time evaluation of an *output of interest* (defined as a functional of the state variable [16]) for many parameter entries, i.e., for many configurations of the problem.

The continuously growing availability of computational power and the simultaneous algorithmic improvements make possible 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 [53]. However, these schemes remain prohibitevely expensive in many-query and real-time contexts, both in terms of CPU time and memory demand. This follows from the large amount of degrees of freedom (DOFs) they imply, resulting from the (fine) spatial discretization needed to accurately solve the underpinning PDE [1]. In light of this, *reduced order modeling* (ROM) methods have received 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, thus to decrease the computational burden while leading to a reasonable loss of accuracy [12].

Reduced basis (RB) methods constitute a well-known and widely-used instance of reduced order modeling techniques. They are generally implemented pursuing an offline-online paradigm [41]. 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 capable of representing the main dynamics of the full-order model [2, 12]. For this, two major approaches have been proposed in the literature: proper orthogonal

Email addresses: `jan.hesthaven@epfl.ch` (J.S. Hesthaven), `stefano.ubbiali@epfl.ch` (S. Ubbiali)

decomposition (POD) [60] and greedy algorithms [31]. The former relies on a deterministic or random sample to generate the snapshots and then employs a singular value decomposition (SVD) to recover the reduced basis. Whereas, the latter adopt a different strategy, as 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 [30]. However, there may 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].

When a reduced-order environment has been properly set up, given a new parameter input, an approximation to the *truth* solution is sought *online* as a linear combination of the RB functions. The expansion coefficients are determined via 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, for complex nonlinear problems with a non-affine dependence on the parameters, projection-based RB methods do not provide any computational gain with respect to a direct (expensive) approach, as the cost to compute the projection coefficients 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 [53]. 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, 49].

A valuable alternative to address this concern is represented by *non-intrusive* RB methods, which refer to the high-fidelity model solely to generate the snapshots, without involving it in the projection process [12]. The projection coefficients are then obtained via interpolation over the parameter domain of a database of reduced-order information [9]. However, it should be noted that reduced bases generally belong to nonlinear, matrix manifolds. Hence, unless using a large amount of samples, standard interpolation techniques may fail, as they cannot enforce the constraints characterizing those manifolds [1, 4].

In this work, we develop a non-intrusive RB method employing POD for the generation of the reduced basis and resorting 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 its accuracy and efficiency, the POD-NN method is applied to the linear and nonlinear Poisson equation and to the steady uncompressible Navier-Stokes equations. Both physical and geometrical parametrizations are considered.

TODO: paper outline

2. Artifical neural networks

Inspired by the biological information processing system (see, e.g., [26, 38]), an *artificial neural network* (ANN), usually simply referred to as *neural network*, is a computational model capable to learn from observational data, i.e., by example, thus providing an alternative to the algorithmic programming paradigm [48]. As its original counterpart, it consists of a collection of processing units, called (artificial) *neurons*, and 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* to all intents and purposes, 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 [59].

Formally, a neural network could be defined as follows [38].

Definition 2.1 (Neural network). *A* neural network *is a sorted triple* $(\mathcal{N}, \mathcal{V}, w)$*, where* $\mathcal{N}$ *is the set of* neurons*, with cardinality* $|\mathcal{N}|$, $\mathcal{V} = \{(i, j), 1 \leq i, j \leq |\mathcal{N}|\}$ *is the set of* connections*, with* (i, j) *denoting the oriented connection linking the sending neuron* i *with the target neuron* j*, and* $w : \mathcal{V} \to \mathbb{R}$ *is the* weight function*, defining the weight* $w_{i,j} = w((i, j))$ *of the connection* (i, j)*. A weight may be either positive or negative, making the underlying connection either excitatory or inhibitory, respectively. By convention,* $w_{i,j} = 0$ *means that neurons* i *and* j *are not directly connected.*

In the following, we dive deeper into the structure and training of a neural network, starting by detailing the structure of an artificial neuron.

2.1. Neuronal model

An artificial neuron represents a simplified model of a biological neuron [38]. To introduce the components of the model, let us consider the neuron j represented in Fig. 2.1. 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 $y_\Omega(t) \in \mathbb{R}$ the scalar output fired by a generic neuron Ω at time t, 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 $w_{j,r_i}\, y_j(t + \Delta t)$. Please note that in the context of ANNs the 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, thus to lighten the notation.

An artificial neuron j is completely characterized by three functions: the propagation function, the activation function, and the output function. These will be defined and detailed below in the same order they get involved in the data flow.

Propagation 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:

$$f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) = \sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} \, . \tag{2.1}$$

The function (2.1) provides a simple, yet effective way of modeling the accumulation of different input electric signals within a biological neuron [26].

Activation or transfer function. 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 $\theta_j \in \mathbb{R}$ [38]:

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

where we have employed the weighted sum (2.1) as propagation function. 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 Fig. 2.2, θ_j is now treated as a synaptic weight, while the neuron threshold is set to zero. Hence, the net input becomes

$$u_j = \sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} - \theta_j \, , \tag{2.2}$$

i.e., the threshold is included in the propagation function rather than in the activation function, which we can now express in the form

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

Conversely to the propagation function, there exist various choices for the activation function. Sigmoid functions have been widely used for the realization of artificial neural networks due to their graceful combination of linear and nonlinear behaviour [26]. Sigmoid functions are s-shaped, monotically increasing, and assume values in a bounded interval. A well-known instance is given by the hyperbolic tangent,

$$f_{act}(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}} \, . \tag{2.3}$$

Output function. 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:

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

Typically, f_{out} is the identity function, so that activation and output of a neuron coincides, i.e., $y_j = f_{out}(a_j) = a_j$. Note that while the input $\mathbf{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ of the neuron is generally vectorial, i.e., $m > 1$, the output is scalar. 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.

The neural model presented so far refers to the so called *computing* neuron, i.e., a neuron which process 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 activation pattern (input vector), without performing any computation [26].

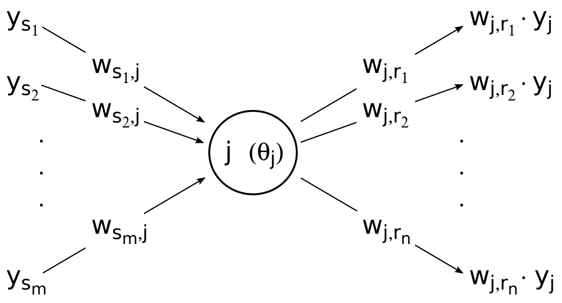

Figure 2.1. Visualization of the generic j-th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j}\ y_{s_1}, \dots, w_{s_m,j}\ y_{s_m}\}$ coming from the sending neurons $s_1, \dots, s_m$, and fires y_j, sent to the target neurons $\{r_1, \dots, r_n\}$ through the synapsis $\{w_{j,r_1}, \dots, w_{j,r_n}\}$. The neuron threshold θ_j is reported within its body.

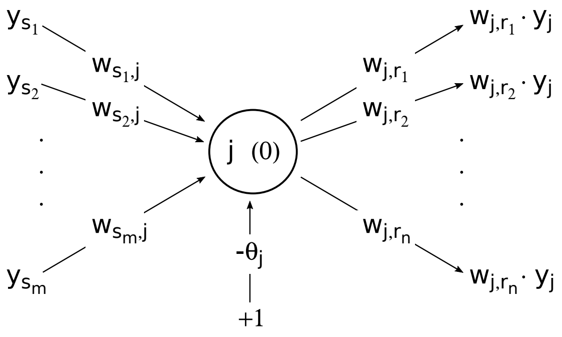

Figure 2.2. Visualization of the generic j-th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j}\ y_{s_1}, \dots, w_{s_m,j}\ y_{s_m}, -\theta_j\}$ coming from the sending neurons $s_1, \dots, s_m$, b, respectively, with b the bias neuron. The neuron output y_j is then conveyed towards the target neurons $\{r_1, \dots, r_n\}$ through the synapsis $\{w_{j,r_1}, \dots, w_{j,r_n}\}$. Observe that, conversely to the model offered in Fig. 2.1, the neuron threshold is now set to 0.

2.2. Network topology: the feedforward neural network

The way neurons are interconnected 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* [56], have been typically 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, and not within the same layer. Then, an *activation pattern* $\mathbf{p} \in \mathbb{R}^{M_I}$, supplied to the network through the source nodes in the first layer, provides the input signal for the neurons in the first hidden layer. For each hidden layer, its output signals give the input pattern for the following layer. In this way, information travels towards the last layer of the network, i.e., 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 makes 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 quite simple structure, the range of application of SLPs is rather limited. Indeed, only *linearly separable* data can be properly represented using SLPs [38]. Conversely, multi-layer perceptrons (MLPs), with at least one hidden layer, are universal function approximators, as stated by Cybenko [13, 14]. In detail:

(i) an MLP with one layer of *hidden neurons* and differentiable activation functions can approximate any *continuous* function [14];

(ii) an MLP with two layers of *hidden neurons* and differentiable activation functions can approximate *any function* [13].

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.

An instance of a three-layer (i.e., two hidden layer plus the output layer) feedforward network is offered in Fig. 2.3. 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.

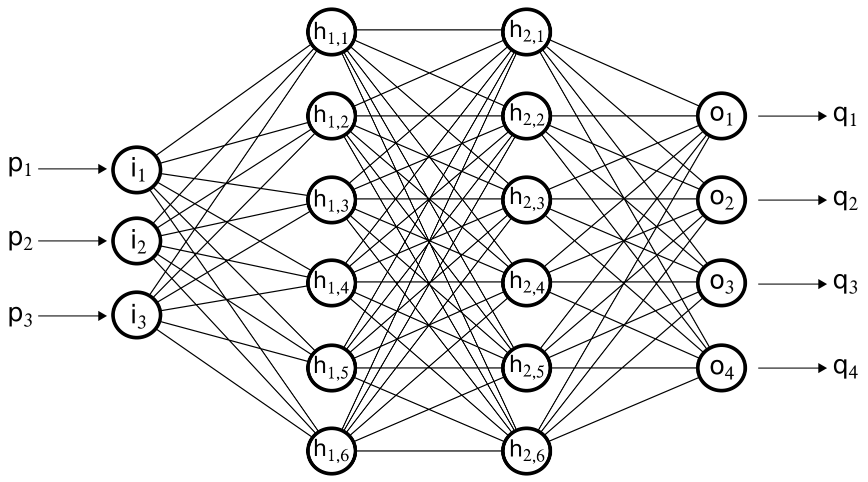

Figure 2.3. A three-layer feedforward neural network, with 3 input neurons, two hidden layers each one consisting of 6 neurons, and 4 output neurons. Within each connection, information flows from left to right.

2.3. 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 by storing the acquired knowledge via the network 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 patterns*. 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 successfull training, the

[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 the output y_j coincides with the corresponding entry of $\mathbf{q}$, i.e., $q_j = y_j$, $j = 1, \ldots, M_O$.

neural network provides reasonable responses for unknown problems of the same class of the training set. This property is known as *generalization* [38].

Training algorithms can be classified based on the nature of the training set, i.e., the set of training patterns. We can then distinguish three *learning paradigms*: supervised learning, unsupervised learning and reinforcement learning [24]. The choice of the learning paradigm is task-dependent. In particular, for function approximation, the *supervised* learning paradigm is the best choice.

Consider the nonlinear unknown function $\boldsymbol{f}$,

$$\begin{aligned} \boldsymbol{f} \, : \, & \mathbb{R}^{M_I} \to \mathbb{R}^{M_O} \\ & \mathbf{x} \quad \mapsto \mathbf{y} = \boldsymbol{f}(\mathbf{x}) \, , \end{aligned}$$

and a set of labeled examples $\{\mathbf{p}_i, \mathbf{t}_i = \boldsymbol{f}(\mathbf{p}_i)\}_{i=1}^{N_{tr}}$ which form the training set. Note that at each *input pattern* $\mathbf{p}_i \in \mathbb{R}^{M_I}$, $i = 1, \dots, N_{tr}$, corresponds an associated desired output or *teaching input* $\mathbf{t}_i \in \mathbb{R}^{M_O}$. 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 $\mathbb{R}^{M_O}$. To this end, the synaptic and bias weights are iteratively adjusted according to a set of well-defined rules. 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. In other terms, the bigger the output of the performance function is, the farther the neural network representation of the world is from the actual reality. Therefore, every learning rule aims to *minimize* the performance E. For this purpose, E should be a scalar function of the free parameters, i.e., the weights, of the network, namely

$$E = E(\mathbf{w}) \in \mathbb{R} \, . \tag{2.4}$$

with $\mathbf{w} \in \mathbb{R}^{|\mathscr{V}|}$ be the vector collecting the weights $\{w_{i,j} = w((i,j))\}_{(i,j) \in \mathscr{V}}$. 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)

$$E(\mathbf{w}) = \sum_{\mathbf{p} \in P} E_{\mathbf{p}}(\mathbf{w}) = \sum_{\mathbf{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\mathbf{p},j} - q_{\mathbf{p},j} \right)^2 \, , \tag{2.5}$$

where P is the training set, the subscript $\mathbf{p}$ refers to the input pattern, $\mathbf{t}$ is the teaching input and $\mathbf{q}$ the actual output. Note that (2.5) accounts for the error committed on each input pattern in the training set, leading to a so-called *offline* learning procedure.

The steps performed by a generic offline supervised training process are listed by Algorithm 2.1. In our numerical tests, at each epoch t the weight update $\Delta \mathbf{w}(t)$ is computed via the Levenberg-Marquardt algorithm, derived in the following subsection.

Algorithm 2.1 Supervised learning algorithm; the procedure to compute the accumulated error is also provided.

```
Input: neural network $(\mathcal{N}, \mathcal{V}, \mathbf{w}_0)$, training set $P = \{\mathbf{p}_i, \mathbf{t}_i\}_{i=1}^{N_{tr}}$,
       metric $d(\cdot,\cdot) : \mathbb{R}^{M_O} \times \mathbb{R}^{M_O} \to \mathbb{R}$, tolerance $\epsilon$, maximum number of epochs $T$
Output: trained neural network $(\mathcal{N}, \mathcal{V}, \mathbf{w}_{opt})$
1:  $t = 0$
2:  $\mathbf{w}(0) = \mathbf{w}_0$
3:  $E(\mathbf{w}(0))$ = AccumulatedError($\mathcal{N}, \mathcal{V}, \mathbf{w}(0), P, d$)
4:  while $t < T$ and $E(\mathbf{w}(t)) > \epsilon$ do
5:      compute weight update $\Delta\mathbf{w}(t)$ based on $E(\mathbf{w}(t))$
6:      $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$
7:      $E(\mathbf{w}(t+1))$ = AccumulatedError($\mathcal{N}, \mathcal{V}, \mathbf{w}(t+1), P, d$)
8:      $t \leftarrow t+1$
9:  end while
10: $\mathbf{w}_{opt} = \mathbf{w}(t)$
11: function $E(\mathbf{w})$ = AccumulatedError($\mathcal{N}, \mathcal{V}, \mathbf{w}, P, d$)
12:     $E(\mathbf{w}) = 0$
13:     for $i = 1, \dots, N_{tr}$ do
14:         evaluate output vector $\mathbf{y}_{\mathbf{p}_i}$, corresponding to input pattern $\mathbf{p}_i$
15:         $E(\mathbf{w}) \leftarrow E(\mathbf{w}) + d(\mathbf{y}_{\mathbf{p}_i}, \mathbf{t}_i)$
16:     end for
17: end function
```

2.3.1. Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm [42] stand as an approximation to Newton's method [23]. If one applies Newton's method for the minimization of the loss function $E = E(\mathbf{w})$, at each iteration the *search direction* $\Delta\mathbf{w}$ is sought by solving the following linear system:

$$\nabla^2 E(\mathbf{w})\ \Delta\mathbf{w} = -\nabla E(\mathbf{w})\,, \tag{2.6}$$

where $\nabla E(\mathbf{w})$ and $\nabla^2 E(\mathbf{w})$ denotes, respectively, the gradient vector and the Hessian matrix of E with respect to its argument $\mathbf{w}$. Assume that the loss function is represented as the accumulated MSE and let us denote by $\mathbf{e_p}$ the error vector corresponding to the input pattern $\mathbf{p}$, namely, $\mathbf{e_p} = \mathbf{t_p} - \mathbf{q_p}$. Introducing the Jacobian $J_\mathbf{p}$ of the specific error vector $\mathbf{e_p}$ with respect to $\mathbf{w}$, i.e.,

$$J_\mathbf{p}(\mathbf{w}) = \begin{bmatrix} \dfrac{\partial e_{\mathbf{p},1}}{\partial w_1} & \cdots & \dfrac{\partial e_{\mathbf{p},1}}{\partial w_{|\mathcal{V}|}} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial e_{\mathbf{p},M_O}}{\partial w_1} & \cdots & \dfrac{\partial e_{\mathbf{p},M_O}}{\partial w_{|\mathcal{V}|}} \end{bmatrix} \in \mathbb{R}^{M_O \times |\mathcal{V}|} \tag{2.7}$$

simple computations yield:

$$\nabla E(\mathbf{w}) = \sum_{\mathbf{p}\in P} \frac{2}{M_0} J_\mathbf{p}(\mathbf{w})^T \mathbf{e_p} \in \mathbb{R}^{|\mathcal{V}|} \tag{2.8}$$

and

$$\nabla^2 E(\mathbf{w}) = \sum_{\mathbf{p}\in P} \frac{2}{M_O} \left[J_\mathbf{p}(\mathbf{w})^T J_\mathbf{p}(\mathbf{w}) + S(\mathbf{w}) \right] \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}\,, \tag{2.9}$$

with

$$S(\mathbf{w}) = \sum_{\mathbf{p}\in P} \frac{2}{M_O} \sum_{j=1}^{M_O} e_{\mathbf{p},j} \nabla^2 e_{\mathbf{p},j}\,.$$

Assuming $S(\mathbf{w}) \approx 0$, inserting (2.8) and (2.9) into (2.6) we get the linear system

$$\left[\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w})\right] \Delta\mathbf{w} = -\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}}(\mathbf{w}). \tag{2.10}$$

The Levenberg-Marquardt modification to Newton's method reads [23, 42]:

$$\left[\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w}) + \mu I\right] \Delta\mathbf{w} = -\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}}(\mathbf{w}), \tag{2.11}$$

with $\mu \geq 0$ and I the identity matrix of size $|\mathcal{V}| \times |\mathcal{V}|$. Note that if $\mu = 0$ we recover Newton's method (2.10), while for $\mu \gg 1$ the search direction $\Delta\mathbf{w}$ approaches the antigradient of E. Hence, the Levenberg-Marquardt algorithm can be seen as an interpolation between Newton's method and the steepest descent method, aiming to retain the advantages of both techniques [23].

The Levenberg-Marquardt training algorithm proceeds as follows. At each epoch t of the training procedure, we solve the (potentially large) linear system (2.11). Whenever the step $\Delta\mathbf{w}(t)$ leads to a reduction in the performance function, i.e., $E(\mathbf{w}(t) + \Delta\mathbf{w}(t)) < E(\mathbf{w}(t))$, the parameter μ is reduced by a factor $\beta > 1$. Conversely, if $E(\mathbf{w}(t) + \Delta\mathbf{w}(t)) > E(\mathbf{w}(t))$ the parameter is multiplied by the same factor β. This reflects the idea that far from the actual minimum we should prefer the gradient method to Newton's method, since the latter may diverge. Yet, once in a neighborhood of the minimum, we switch to Newton's method so to exploit its faster convergence [42].

The key step in the algorithm is the computation of the Jacobian $J_{\mathbf{p}}(\mathbf{w})$ for each training vector $\mathbf{p}$. Suppose that the k-th element w_k of $\mathbf{w}$ represents the weight $w_{i,j}$ of the connection (i, j), for some i and j, with $1 \leq i, j \leq |\mathcal{N}|$, i.e., $w_k = w_{i,j}$. Then:

$$\frac{\partial e_{\mathbf{p},h}}{\partial w_{i,j}} = \frac{\partial e_{\mathbf{p},h}}{\partial u_{\mathbf{p},j}} \frac{\partial u_{\mathbf{p},j}}{\partial w_{i,j}} = \frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} \frac{\partial y_{\mathbf{p},j}}{\partial u_{\mathbf{p},j}} \frac{\partial u_{\mathbf{p},j}}{\partial w_{i,j}} = \frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} f'_{act}(u_{\mathbf{p},j})\, y_{\mathbf{p},i} = \delta_{\mathbf{p},h,j}\, y_{\mathbf{p},i}, \tag{2.12}$$

with

$$\delta_{\mathbf{p},h,j} := -\frac{\partial e_{\mathbf{p},h}}{\partial u_{\mathbf{p},j}} = -\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} f'_{act}(u_{\mathbf{p},j}). \tag{2.13}$$

For the computation of the derivative

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}}$$

occurring in (2.13), we assume that, within the set of neurons $\mathcal{N}$, items are ordered such that the output neurons come first, i.e.,

$$j \text{ output neuron} \quad \Leftrightarrow \quad 1 \leq j \leq M_O.$$

We can then distinguish three cases:

(i) j output neuron, $j = h$: since $e_{\mathbf{p},h} = e_{\mathbf{p},j} = t_{\mathbf{p},j} - y_{\mathbf{p},j}$, then

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} = -1;$$

(ii) j output neuron, $j \neq h$: the output of an output neuron can not influence the signal fired by another output neuron, hence

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} = 0;$$

(iii) j inner neuron: letting R be the set of successors of j, the chain rule yields

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} = \sum_{r\in R} \frac{\partial e_{\mathbf{p},h}}{\partial u_{\mathbf{p},r}} \frac{\partial u_{\mathbf{p},r}}{\partial y_{\mathbf{p},j}} = -\sum_{r\in R} \delta_{\mathbf{p},h,r}\, w_{j,r}.$$

Ultimately, at any iteration of the learning algorithm, the entries of the Jacobian matrix $J_{\mathbf{p}}$ are given by

$$\frac{\partial e_{\mathbf{p},h}}{\partial w_k} = \delta_{\mathbf{p},h,j}\; y_{\mathbf{p},i}\,, \quad 1 \le h \le M_O\,, 1 \le k \le |\mathscr{V}|\,, w_k = w_{i,j} \text{ for some } i \text{ and } j\,,$$

with

$$\delta_{\mathbf{p},h,j} = \begin{cases} f'_{act}(u_{\mathbf{p},j}) \sum_{r \in R} \delta_{\mathbf{p},h,r}\; w_{j,r}\,, & \text{if } j \text{ inner neuron }, \quad (2.14a) \\ f'_{act}(u_{\mathbf{p},j})\; \delta^K_{jh}\,, & \text{if } j \text{ output neuron }, \quad (2.14b) \end{cases}$$

where δ^K_{jh} is the Kronecker delta. Observe that (2.14a) defines $\delta_{\mathbf{p},h,j}$ for a hidden node j by relying on the neurons in the following layer, whereas (2.14b) only involves the net input $u_{\mathbf{p},j}$ of the neuron itself. Therefore, the coupled equations (2.14a)-(2.14b) implicitly set the order in which the entries of $J_{\mathbf{p}}(\mathbf{w})$ have to be determined: starting from the output layer, compute the terms related to connections ending in that layer, then move backwards to the preceeding layer. In this way, the error is *backpropagated* from the output down to the input, leaving traces in each layer of the network [38, 61].

Let us finally remark that a trial and error approach is required to find satisfactory values for μ and β; as proposed in [42], a good starting point may be $\mu = 0.01$, with $\beta = 10$. Moreover, the dimension of the system (2.11) increases nonlinearly with the number of neurons in the network, making the Levenberg-Marquardt algorithm poorly efficient for large networks [23]. However, it turns out to be efficient and accurate for networks with few hundreads of connections.

3. Parametrized partial differential equations

Assume $\mathscr{P}_{ph} \subset \mathbb{R}^{P_{ph}}$ and $\mathscr{P}_g \subset \mathbb{R}^{P_g}$ be compact sets, and let $\boldsymbol{\mu}_{ph} \in \mathscr{P}_{ph}$ and $\boldsymbol{\mu}_g \in \mathscr{P}_g$ be respectively the *physical* and *geometrical* parameters characterizing the differential problem, so that $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P} = \mathscr{P}_{ph} \times \mathscr{P}_g \subset \mathbb{R}^P$, $P = P_{ph} + P_g$ 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 $\widetilde{\Omega} = \widetilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 1, 2$, with Lipschitz boundary $\widetilde{\Gamma} = \partial\widetilde{\Omega}$; we denote by $\widetilde{\Gamma}_D$ and $\widetilde{\Gamma}_N$ the portions of $\widetilde{\Gamma}$ where Dirichlet and Neumann boundary conditions are enforced, with $\widetilde{\Gamma}_D \cup \widetilde{\Gamma}_N = \widetilde{\Gamma}$ and $\mathring{\widetilde{\Gamma}}_D \cap \mathring{\widetilde{\Gamma}}_N = \emptyset$.

Consider then 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 of $\widetilde{V}$, i.e., the space of linear and continuous functionals over $\widetilde{V}$. Denoting by $\widetilde{G} : \widetilde{V} \times \mathscr{P}_{ph} \to \widetilde{V}'$ the map representing a parametrized nonlinear second-order PDE, the differential (strong) form of the problem of interest reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{G}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) = 0 \quad \text{in } \widetilde{V}'(\boldsymbol{\mu}_g)\,, \tag{3.1}$$

namely

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

Here, $\langle \cdot,\cdot \rangle_{\widetilde{V}',\widetilde{V}} : \widetilde{V}' \times \widetilde{V} \to \mathbb{R}$ represents the duality pairing between $\widetilde{V}'$ and $\widetilde{V}$, encoding the action of any functional of $\widetilde{V}'$ onto elements of $\widetilde{V}$.

The finite element method requires problem (3.1) to be stated in a weak (or variational) form [52]. To this end, let us introduce the form $\widetilde{g} : \widetilde{V} \times \widetilde{V} \times \mathscr{P} \to \mathbb{R}$, with $g(\cdot,\cdot;\boldsymbol{\mu})$ 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 (3.1) then reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{g}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = 0 \quad \forall v \in \widetilde{V}(\boldsymbol{\mu})\,. \tag{3.2}$$

Note that the definition of $\widetilde{g}(\cdot,\cdot;\boldsymbol{\mu})$ relies on the duality pairing $\langle \cdot,\cdot \rangle_{\widetilde{V}',\widetilde{V}}$ between $\widetilde{V}$ and $\widetilde{V}'$. Hence, from the nonlinearity of $\widetilde{G}$ follows the nonlinearity of $\widetilde{g}$ with respect to its first argument.

3.1. *From physical to reference domain*

As anticipated in the Introduction, any reduced basis method seeks an approximated solution to a differential problem as a combination of (few) well-chosen basis vectors, resulting in a finite-dimensional model which features a remarkably decreased dimension with respect to canonical discretization techniques (e.g., finite element method). As further detailed in Section 5, the method relies on the combination of a collection of high-fidelity approximations $\{\widetilde{u}_h(\boldsymbol{\mu}^{(1)}), \dots, \widetilde{u}_h(\boldsymbol{\mu}^{(N)})\}$, called *snapshots*, corresponding to the parameter values $\{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$. However, we may be concerned with boundary value problems for stationary PDEs defined on variable shape geometries, so that two snapshots $\widetilde{u}_h(\boldsymbol{\mu}^{(j)})$ and $\widetilde{u}_h(\boldsymbol{\mu}^{(k)})$, with $1 \leq j, k \leq N$ and $j \neq k$, are likely to have been computed on two different domains $\widetilde{\Omega}(\boldsymbol{\mu}_g^{(j)})$ and $\widetilde{\Omega}(\boldsymbol{\mu}_g^{(k)})$. Hence, the underlying high-fidelity solver should be carefully designed to guarantee the *compatibility* among snapshots. In particular, let $\mathbf{u}_h(\boldsymbol{\mu}^{(j)})$ and $\mathbf{u}_h(\boldsymbol{\mu}^{(k)})$ be the vectors collecting the degrees of freedom for $\widetilde{u}_h(\boldsymbol{\mu}^{(j)})$ and $\widetilde{u}_h(\boldsymbol{\mu}^{(j)})$, respectively. Then, we should ensure that:

(a) $\dim(\mathbf{u}_h(\boldsymbol{\mu}^{(j)})) = \dim(\mathbf{u}_h(\boldsymbol{\mu}^{(k)}))$, i.e. the number of degrees of freedom must be the same;

(b) corresponding entries of the two vectors must be correlated.

For a mesh-based numerical method, the conditions (a) and (b) can be satisfied by preserving the connectivity of the underlying meshes across different domains, i.e., different values of $\boldsymbol{\mu}_g$. To this end, we formulate and solve the differential problem over a fixed, *parameter-independent* domain Ω. This can be accomplished upon introducing a parametrized map $\mathbf{\Phi} : \Omega \times \mathscr{P}_g \to \widetilde{\Omega}$ such that

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

The transformation $\mathbf{\Phi}$ allows to restate the general problem (3.1). Let V be a suitable Hilbert space over Ω and V' be its dual. Suppose V is equipped with the scalar product $(\cdot,\cdot)_V : V \times V \to \mathbb{R}$ and the induced norm $\|\cdot\|_V = \sqrt{(\cdot,\cdot)_V} : V \to [0,\infty)$. Given the parametrized map $G : V \times \mathscr{P} \to V'$ representing the (nonlinear) PDE over the reference domain Ω, we focus on differential problems of the form: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u(\boldsymbol{\mu}) \in V$ such that

$$G(u(\boldsymbol{\mu}); \boldsymbol{\mu}) = 0 \quad \text{in } V'. \tag{3.3}$$

The weak formulation of problem (3.3) reads: given $\boldsymbol{\mu} \in \mathscr{P}$, seek $u(\boldsymbol{\mu}) \in V$ such that

$$g(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = 0 \quad \forall v \in V, \tag{3.4}$$

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

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

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 $\mathbf{\Phi}$, 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

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

For any $\boldsymbol{\mu} \in \mathscr{P}$ we seek a discrete solution $u_h(\boldsymbol{\mu})$ to the problem (3.3) on a *parameter-independent* cover Ω_h of the domain Ω. Provided a convenient choice for Ω, this makes the mesh generation process easier. In addition, we note that discretizing the problem (3.3) over Ω_h is equivalent to approximating the original problem (3.1) over the mesh $\widetilde{\Omega}_h(\boldsymbol{\mu}_g)$, given by

$$\widetilde{\Omega}_h(\boldsymbol{\mu}_g) = \mathbf{\Phi}(\Omega_h; \boldsymbol{\mu}_g).$$

Therefore, the requirements (a) and (b) are automatically fulfilled provided the map $\mathbf{\Phi}(\cdot; \boldsymbol{\mu}_g)$ is *conformal* for any $\boldsymbol{\mu}_g \in \mathscr{P}_g$. To ensure conformality, in our numerical tests we resort to a particular choice for $\mathbf{\Phi}$ - the boundary displacement-dependent transfinite map (BDD TM) proposed in [34].

3.2. *Problems of interest*

Within the wide range of PDEs which suit the functional framework portrayed so far, in this work we focus on two relevant examples - the nonlinear Poisson equation and the stationary Navier-Stokes equations - which will serve as test cases in Section **??**.

3.2.1. 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 [44]. We consider the following version of the parametrized Poisson equation for a state variable $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$:

$$
\begin{cases}
-\widetilde{\nabla} \cdot \left(\widetilde{k}(\widetilde{\boldsymbol{x}}, \widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) \, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \right) = \widetilde{s}(\widetilde{\boldsymbol{x}}; \boldsymbol{\mu}_{ph}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g) , \quad (3.5a) \\
\widetilde{u}(\boldsymbol{\mu}) = \widetilde{g}(\widetilde{\boldsymbol{\sigma}}; \boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}_D , \quad (3.5b) \\
\widetilde{k}(\widetilde{\boldsymbol{\sigma}}, \widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) \, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\boldsymbol{n}} = 0 & \text{on } \widetilde{\Gamma}_N . \quad (3.5c)
\end{cases}
$$

Here, for any $\boldsymbol{\mu}_g \in \mathscr{P}_g$:

- $\widetilde{\boldsymbol{x}}$ 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 $\widetilde{\boldsymbol{\sigma}}$;
- $\widetilde{k} \, : \, \widetilde{\Omega} \times \mathbb{R} \times \mathscr{P}_{ph} \to (0, \infty)$ is the diffusion coefficient, $\widetilde{s} \, : \, \widetilde{\Omega} \times \mathscr{P}_{ph} \to \mathbb{R}$ is the source term, and $\widetilde{g} \, : \, \widetilde{\Gamma}_D \times \mathscr{P}_{ph} \to \mathbb{R}$ encodes the Dirichlet boundary conditions;
- to ease the subsequent discussion, we limit the attention to homogeneous Neumann boundary constraints.

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

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

Multiplying (3.5a) by a *test* function $v \in \widetilde{V}$, integrating over $\widetilde{\Omega}$, and exploiting integration by parts on the left-hand side, yields:

$$
\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) \, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) = \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph}) \; v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) , \tag{3.6}
$$

where we have omitted the dependence on the space variable $\widetilde{\boldsymbol{x}}$ for ease of notation. For the integrals in Eq. (3.6) to be well-defined, we require, for any $\boldsymbol{\mu}_g \in \mathscr{P}_g$,

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

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

$$
\begin{aligned}
\widetilde{a}(w, v; \boldsymbol{\mu}) := & \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + \widetilde{l}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) \, \widetilde{\nabla} w \cdot \widetilde{\nabla} v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) \\
& + \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + \widetilde{l}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) \, \widetilde{\nabla} \widetilde{l}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \, w, v \in \widetilde{V}(\boldsymbol{\mu}_g) ,
\end{aligned} \tag{3.7a}
$$

$$
\widetilde{f}(v; \boldsymbol{\mu}) := \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph}) \; v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \, v \in \widetilde{V}(\boldsymbol{\mu}_g) , \tag{3.7b}
$$

the weak formulation of problem (3.5) reads: given $\boldsymbol{\mu} \in \mathscr{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$
\widetilde{a}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = \widetilde{f}(v; \boldsymbol{\mu}) \quad \forall \, v \in \widetilde{V}(\boldsymbol{\mu}_g) , \tag{3.8}
$$

Then, the weak solution of problem (3.5) is given by $\widetilde{u}(\boldsymbol{\mu}) + \widetilde{l}(\boldsymbol{\mu})$. Note that using a lifting function makes the formulation (3.8) *symmetric*, i.e., both the solution and the test functions are picked up from the same functional space [52].

Let us now re-state the variational problem (3.8) 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 $\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})$ the Jacobian of the parametrized map $\boldsymbol{\Phi}(\cdot;\boldsymbol{\mu})$, with determinant $|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|$, and we set

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

Letting

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

and exploiting standard change of variables formulas, the variational formulation of the Poisson problem (3.5) over Ω reads: given $\boldsymbol{\mu}\in\mathscr{P}$, find $u(\boldsymbol{\mu})\in V$ such that

$$a(u(\boldsymbol{\mu}),\,v;\boldsymbol{\mu}) = f(v;\boldsymbol{\mu}) \quad \forall\, v\in V, \tag{3.9}$$

with

$$\begin{aligned} a(w,\,v;\boldsymbol{\mu}) = & \int_\Omega k(w+l(\boldsymbol{\mu});\boldsymbol{\mu})\ \mathbb{J}^{-T}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\nabla w\cdot \mathbb{J}^{-T}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\nabla v\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\,|\,d\Omega \\ & + \int_\Omega k(w+l(\boldsymbol{\mu});\boldsymbol{\mu})\ \mathbb{J}^{-T}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\nabla l(\boldsymbol{\mu})\cdot \mathbb{J}^{-T}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\nabla v\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\,|\,d\Omega, \end{aligned} \tag{3.10a}$$

$$f(v;\boldsymbol{\mu}) = \int_\Omega s(\boldsymbol{\mu})\ v\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,d\Omega, \tag{3.10b}$$

for any $w,\,v\in V$ and $\boldsymbol{\mu}\in\mathscr{P}$. Note that, as done in (3.7), we resort to a lifting function $l(\boldsymbol{\mu})\in H^1(\Omega)$ with $l(\boldsymbol{\mu})\big|_{\Gamma_D} = g(\boldsymbol{\mu})$, such that the weak solution to problem (3.5) re-stated over Ω is obtained as $u(\boldsymbol{\mu}) + l(\boldsymbol{\mu})$.

Let us also remark that, when on a parameter-independent configuration, one can avoid to distinguish between physical and geometrical parameters, since even the latter now affect the integrands in (3.10a) and (3.10b), and not the domain of integration.

3.2.2. Steady uncompressible 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 region $\widetilde{\Omega}(\boldsymbol{\mu}_g)\subset\mathbb{R}^d$, $d=2,3$ [54]. Letting $\widetilde{\boldsymbol{v}} = \widetilde{\boldsymbol{v}}(\widetilde{\boldsymbol{x}};\boldsymbol{\mu})$ and $\widetilde{p} = \widetilde{p}(\widetilde{\boldsymbol{x}};\boldsymbol{\mu})$ be the velocity and pressure of the fluid, respectively, the parametrized steady version of the Navier-Stokes equations considered in this work reads:

$$\begin{cases} \widetilde{\nabla}\cdot\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = 0 & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g), \quad (3.11a) \\ -\nu(\boldsymbol{\mu})\,\widetilde{\Delta}\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + (\widetilde{\boldsymbol{v}}(\boldsymbol{\mu})\cdot\widetilde{\nabla})\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \dfrac{1}{\rho(\boldsymbol{\mu})}\widetilde{\nabla}\widetilde{p}(\boldsymbol{\mu}) = \boldsymbol{0} & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g), \quad (3.11b) \\ \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = \widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}_D(\boldsymbol{\mu}_g), \quad (3.11c) \\ \widetilde{p}(\boldsymbol{\mu})\widetilde{\boldsymbol{n}} - \nu(\boldsymbol{\mu})\widetilde{\nabla}\widetilde{\boldsymbol{v}}(\boldsymbol{\mu})\cdot\widetilde{\boldsymbol{n}} = \boldsymbol{0} & \text{on } \widetilde{\Gamma}_N(\boldsymbol{\mu}_g). \quad (3.11d) \end{cases}$$

Here, $\widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph})$ denotes the velocity field prescribed on $\widetilde{\Gamma}_D$, whereas homogeneous Neumann conditions are applied on $\widetilde{\Gamma}_N$. Furthermore, $\rho(\boldsymbol{\mu})$ and $\nu(\boldsymbol{\mu})$ represent the uniform density and kinematic viscosity of the fluid, respectively. Note that, despite 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 Reynolds number, which combine physical properties of the fluid with geometrical features of the domain. Therefore, a numerical study of the sensitivity of the system (3.11) with respect to $\boldsymbol{\mu}_g$ may be carried out by adapting either $\rho(\boldsymbol{\mu})$ or $\nu(\boldsymbol{\mu})$ as $\boldsymbol{\mu}_g$ varies, so to preserve a dimensionless quantity of interest; we refer the reader to Section **??** for a practical example.
For our purpose, it is worth noticing that the nonlinearity of problem (3.11) lies in the convective term

$$\big(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu})\cdot\widetilde{\nabla}\big)\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}),$$

which gives rise to a *quadratic* nonlinearity.

To write the differential system (3.11) in weak form over a $\boldsymbol{\mu}_g$-independent configuration Ω, let us introduce the velocity and pressure spaces

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

respectively, with

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

be their respective counterparts over Ω. By multiplying (3.11) for test functions $\left(\widetilde{\boldsymbol{\chi}}, \widetilde{\xi}\right) \in \widetilde{X}(\boldsymbol{\mu}_g) \times \widetilde{Q}(\boldsymbol{\mu}_g)$, 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 $\boldsymbol{\mu} \in \mathscr{P}$, find $u(\boldsymbol{\mu}) = (\boldsymbol{v}(\boldsymbol{\mu}), p(\boldsymbol{\mu})) \in V = X \times Q$ so that

$$a(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + d(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + b(p(\boldsymbol{\mu}), \nabla\cdot\boldsymbol{\chi}; \boldsymbol{\mu}) = f_1(\boldsymbol{\chi}; \boldsymbol{\mu}), \tag{3.12a}$$

$$b(\nabla\cdot\boldsymbol{v}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) = f_2(\xi; \boldsymbol{\mu}), \tag{3.12b}$$

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

$$c(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = \int_\Omega \left(\boldsymbol{\psi}\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\right)\boldsymbol{\chi}\cdot\boldsymbol{\eta}\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,d\Omega \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in \left[H^1(\Omega)\right]^d, \tag{3.13a}$$

$$a(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \nu(\boldsymbol{\mu})\int_\Omega \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\boldsymbol{\psi} : \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\boldsymbol{\chi}\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,d\Omega \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in X, \tag{3.13b}$$

$$b(\boldsymbol{\psi}, \xi; \boldsymbol{\mu}) = -\frac{1}{\rho(\boldsymbol{\mu})}\int_\Omega \left(\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\cdot\boldsymbol{\psi}\right)\,\xi\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,d\Omega \qquad \forall \boldsymbol{\psi} \in X, \forall \xi \in Q, \tag{3.13c}$$

$$d(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = c(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{\psi}, \boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}); \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in X, \tag{3.13d}$$

$$f_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = -a(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) - c(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) \qquad \forall \boldsymbol{\psi} \in X, \tag{3.13e}$$

$$f_2(\xi; \boldsymbol{\mu}) = -b(\boldsymbol{l}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) \qquad \forall \xi \in Q. \tag{3.13f}$$

Here, $\boldsymbol{l}(\boldsymbol{\mu}) \in \left[H^1(\Omega)\right]^d$ denotes the lifting vector field, with $\boldsymbol{l}(\boldsymbol{\mu})\big|_{\Gamma_D} = \boldsymbol{g}(\boldsymbol{\mu})$, $\boldsymbol{g}(\boldsymbol{x}; \boldsymbol{\mu}) = \widetilde{\boldsymbol{g}}(\boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\mu}); \boldsymbol{\mu})$ being the velocity field prescribed on Γ_D. Hence, the weak solution to (3.11) defined over the fixed domain Ω is given by $\left(\boldsymbol{v}(\boldsymbol{\mu}) + \boldsymbol{l}(\boldsymbol{\mu}), p(\boldsymbol{\mu})\right)$.

4. Discrete full-order model

Let $V_h \subset V$ be a finite-dimensional subspace of V of dimension M. The FE approximation of the weak problem (3.4) can be cast in the form: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u_h(\boldsymbol{\mu}) \in V_h$ such that

$$g(u_h(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) = 0 \quad \forall v_h \in V_h. \tag{4.1}$$

The discretization (4.1) is known as *Galerkin approximation*, and therefore $u_h(\boldsymbol{\mu})$ is referred to as the *Galerkin solution* to the problem (3.3).

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 (4.1). In this regard, let

$$dg[z](\cdot,\cdot;\boldsymbol{\mu}) : V \times V \to \mathbb{R}$$

be the partial Frechét derivative of $g(\cdot,\cdot;\boldsymbol{\mu})$ with respect to its first argument, evaluated at $z \in V$. Starting from an initial guess $u_h^0(\boldsymbol{\mu})$, we construct a collection of approximations $\left\{u_h^k(\boldsymbol{\mu})\right\}_{k\ge 0}$ to the Galerkin solution $u_h(\boldsymbol{\mu})$ by iteratively solving the linearized problems

$$dg\left[u_h^k(\boldsymbol{\mu})\right](\delta u_h^k(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) = -g(u_h^k(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) \quad \forall v_h \in V_h$$

in the unknown $\delta u_h^k(\boldsymbol{\mu}) \in V_h$, and then setting $u_h^{k+1}(\boldsymbol{\mu}) = u_h^k(\boldsymbol{\mu}) + \delta u_h^k(\boldsymbol{\mu})$.

To derive the algebraic counterpart of the Galerkin-Newton method, let $\{\phi_1, \ldots, \phi_M\}$ be a basis for the M-dimensional space V_h, so that the solution $u_h(\boldsymbol{\mu})$ can be expressed as a linear combination of the basis functions, i.e.,

$$u_h(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu})\,\phi_j(\boldsymbol{x}). \tag{4.2}$$

Hence, denoting by $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$ the vector collecting the *degrees of freedom* $\{u_h^{(j)}\}_{j=1}^M$ and exploiting the linearity of $g(\cdot, \cdot; \boldsymbol{\mu})$ in the second argument, the problem (4.1) is equivalent to: given $\boldsymbol{\mu} \in \mathscr{P}$, find $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$ such that

$$g\left(\sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu})\, \phi_j, \phi_i; \boldsymbol{\mu}\right) = 0 \quad \forall i = 1, \ldots, M.$$

We observe now that the above problem can be written in compact form as

$$\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^M, \tag{4.3}$$

where the i-th component of the *residual vector* $\mathbf{G}(\cdot; \boldsymbol{\mu}) \in \mathbb{R}^M$ is given by

$$\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu})\big)_i = g\left(\sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu})\, \phi_j, \phi_i; \boldsymbol{\mu}\right), \quad i = 1, \ldots, M. \tag{4.4}$$

Then, for $k \geq 0$, the k-th iteration of Newton's method applied to the system (4.3) entails the resolution of the *linear* system

$$\mathbb{J}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big)\, \delta\mathbf{u}_h^k(\boldsymbol{\mu}) = -\mathbf{G}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big), \quad \delta\mathbf{u}_h^k(\boldsymbol{\mu}) \in \mathbb{R}^M, \tag{4.5}$$

so that $\mathbf{u}_h^{k+1}(\boldsymbol{\mu}) = \mathbf{u}_h^k(\boldsymbol{\mu}) + \delta\mathbf{u}_h^k(\boldsymbol{\mu})$. Here, $\mathbb{J}_h(\cdot; \boldsymbol{\mu}) \in \mathbb{R}^{M \times M}$ denotes the Jacobian of the residual vector $\mathbf{G}_h(\cdot; \boldsymbol{\mu})$; exploiting the bilinearity of $dg[z](\cdot, \cdot; \boldsymbol{\mu})$, $\mathbb{J}_h(\cdot; \boldsymbol{\mu})$ is defined as

$$\Big(\mathbb{J}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big)\Big)_{i,j} = dg\big[u_h^k(\boldsymbol{\mu})\big](\phi_j, \phi_i; \boldsymbol{\mu}), \quad i, j = 1, \ldots, M.$$

5. Projection-based reduced basis method

As detailed in the previous section, the finite element discretization of the $\boldsymbol{\mu}$-dependent nonlinear differential problem (3.4), combined with Newton's method, entails the assembly and resolution of (possibly) many linear systems of the form (4.5), whose dimension is directly related to (i) the size of the underlying grid and (ii) the order of the polynomial FE spaces 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 hardly affordable in *many-query* and *real-time* contexts, even resorting to high-performance parallel workstations. This motivates the broad use of *reduced-order* models, across several inter-disciplinary areas, e.g., parameter estimation, optimal control, shape optimization and uncertainty quantification [30, 53].

Reduced basis methods seek an approximated solution to the problem (3.4) as a linear combination of parameter-independent functions

$$\{\psi_1, \ldots, \psi_L\} \subset V_h,$$

called *reduced basis functions*, built from a collection of high-fidelity solutions

$$\{u_h(\boldsymbol{\mu}^{(1)}), \ldots, u_h(\boldsymbol{\mu}^{(N)})\} \subset \mathscr{M}_h,$$

called *snapshots*, where the discrete and finite set

$$\Xi_N = \{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(N)}\} \subset \mathscr{P}$$

may consist of either a uniform lattice or randomly generated points over the parameter domain $\mathscr{P}$ [30]. The basis functions $\{\psi_l\}_{l=1}^L$ 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 $\{\boldsymbol{\mu}^{(n)}\}_{n=1}^N$ must be carefully chosen according to some optimality criterium (see, e.g., [12]). Here, we pursue the first approach, employing the well-known Proper Orthogonal Decomposition (POD) method [60], detailed in the following subsection.

Assume now that a reduced basis is available and let $V_{\mathtt{rb}} \subset V_h$ be the associated *reduced basis space*, i.e.,

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

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

$$u_L(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{l=1}^{L} u_{\mathtt{rb}}^{(l)}(\boldsymbol{\mu})\, \psi_l(\boldsymbol{x}) \ \in \ V_{\mathtt{rb}}, \tag{5.1}$$

with

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

be the *coefficients* (also called *generalized coordinates*) for the expansion of the RB solution in the RB basis functions.

To unearth $u_L(\boldsymbol{\mu})$, we proceed to project the variational problem (3.4) onto the RB space $V_{\mathtt{rb}}$ by pursuing a standard Galerkin approach, leading to the following *reduced basis problem*: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u_L(\boldsymbol{\mu}) \in V_{\mathtt{rb}}$ so that

$$g(u_L(\boldsymbol{\mu}), v_L; \boldsymbol{\mu}) = 0 \quad \forall v_L \in V_{\mathtt{rb}}. \tag{5.2}$$

Then, Newton's method applied to (5.2) entails, at each iteration $k \geq 0$, the solution of the linearized problem: given $\boldsymbol{\mu} \in \mathscr{P}$, seek $\delta u_L^k(\boldsymbol{\mu})$ such that

$$dg\left[u_L^k(\boldsymbol{\mu})\right]\left(\delta u_L^k(\boldsymbol{\mu}), v_L; \boldsymbol{\mu}\right) = -g\left(u_L^k(\boldsymbol{\mu}), v_L; \boldsymbol{\mu}\right) \quad \forall v_L \in V_{\mathtt{rb}},$$

with $u_L^{k+1}(\boldsymbol{\mu}) = u_L^k(\boldsymbol{\mu}) + \delta u_L^k(\boldsymbol{\mu})$.

Let us point out that the RB functions $\{\psi_l\}_{l=1}^L$ belong to V_h, i.e., they are actual FE functions. Hence, denoting by $\boldsymbol{\psi}_l \in \mathbb{R}^M$ the vector collecting the nodal values of ψ_l, for $l = 1, \dots, L$, we introduce the matrix

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

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

$$\mathbf{v}_L = \mathbb{V}\, \mathbf{v}_{\mathtt{rb}}. \tag{5.3}$$

Therefore, each element v_L of the reduced space admits two (algebraic) representations:

- $\mathbf{v}_{\mathtt{rb}} \in \mathbb{R}^L$, collecting the coefficients for the expansion of v_L in terms of the RB basis $\{\psi_1, \dots, \psi_L\}$;
- $\mathbf{v}_L \in \mathbb{R}^M$, collecting the coefficients for the expansion of v_L in terms of the FE basis $\{\phi_1, \dots, \phi_M\}$.

Note that the latter is also available for any $v_h \in V_h$, while the former characterizes the element in the subspace $V_{\mathtt{rb}}$.
Upon choosing $v_L = \psi_l$, $l = 1, \dots, L$, in the RB problem (5.2), for any $\boldsymbol{\mu} \in \mathscr{P}$ we get the set of equations

$$g(u_L(\boldsymbol{\mu}), \psi_l; \boldsymbol{\mu}) = 0 \quad 1 \leq l \leq L. \tag{5.4}$$

Inserting into (5.4) the expansion of ψ_l, $l = 1, \dots, L$, in terms of the canonical FE basis $\{\phi_m\}_{m=1}^M$, i.e.,

$$\psi_l(\boldsymbol{x}) = \sum_{m=1}^{M} \psi_l^{(m)}\, \phi_m(\boldsymbol{x}) = \sum_{m=1}^{M} \mathbb{V}_{m,l}\, \phi_m(\boldsymbol{x}),$$

and exploiting the linearity of $g(\cdot, \cdot; \boldsymbol{\mu})$ in the second argument, yields:

$$0 = \sum_{m=1}^{M} \mathbb{V}_{m,l}\, g(u_L(\boldsymbol{\mu}), \phi_m; \boldsymbol{\mu}) = \left(\mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L(\boldsymbol{\mu}); \boldsymbol{\mu})\right)_l \quad 1 \leq l \leq L,$$

where the last equality follows from the definition (4.4) of the residual vector $\mathbf{G}_h(\cdot; \boldsymbol{\mu})$ and the notation introduced in (5.3).

Then, the algebraic formulation of the reduced basis problem (5.2) can be written in compact form as:

$$\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu});\boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L(\boldsymbol{\mu});\boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu});\boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L. \tag{5.5}$$

This *reduced nonlinear system* imposes the orthogonality (in the Euclidean scalar product) of the residual vector $\mathbf{G}_h(\cdot;\boldsymbol{\mu})$, evaluated in $\mathbb{V}\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu})$, to the columns of $\mathbb{V}$, thus encoding the Galerkin approach pursued at the variational level.
Finally, exploiting the chain rule and the Jacobian $\mathbb{J}_h(\cdot;\boldsymbol{\mu})$ of $\mathbf{G}_h(\cdot;\boldsymbol{\mu})$, the Jacobian $\mathbb{J}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$ of $\mathbf{G}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$ is given by

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{w};\boldsymbol{\mu}) = \mathbb{V}^T\,\mathbb{J}_h(\mathbb{V}\,\mathbf{w};\boldsymbol{\mu})\,\mathbb{V} \in \mathbb{R}^{L\times L} \quad \forall \mathbf{w}\in\mathbb{R}^L,\ \forall\boldsymbol{\mu}\in\mathscr{P}. \tag{5.6}$$

Hence, starting from an initial guess $\mathbf{u}_{\mathtt{rb}}^0 \in \mathbb{R}^L$, each iteration k, $k \geq 0$, of Newton's method applied to the reduced nonlinear system (5.5) entails the resolution of the linear system

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu});\boldsymbol{\mu})\,\delta\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu}) = -\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu});\boldsymbol{\mu}), \tag{5.7}$$

with $\mathbf{u}_{\mathtt{rb}}^{k+1}(\boldsymbol{\mu}) = \mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu}) + \delta\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu})$. Therefore, applying the POD-Galerkin RB method enables a dramatic reduction of the size of the linear systems to solve whenever the dimension L of the reduced space $V_{\mathtt{rb}}$ is much lower than the dimension M of the underlying finite element space V_h.

5.1. Proper Orthogonal Decomposition

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

$$\mathscr{M}_{\Xi_N} = \operatorname{span}\{u_h(\boldsymbol{\mu}^{(1)}),\ldots,u_h(\boldsymbol{\mu}^{(N)})\}.$$

Clearly, $\mathscr{M}_{\Xi_N} \subset \mathscr{M}_h$ and we can assume that $\mathscr{M}_{\Xi_N}$ provides a good approximation of $\mathscr{M}_h$, as long as the number of snapshots is sufficiently large (but typically much smaller than the dimension M of the FE space). Then, we aim at finding a parameter-independent *reduced basis* for $\mathscr{M}_{\Xi_N}$, i.e., a collection of FE functions $\{\psi_1,\ldots,\psi_L\} \subset \mathscr{M}_{\Xi_N}$, with $L \ll M, N$, and L *independent* of both M and N, so that the associated linear space

$$V_{\mathtt{rb}} = \operatorname{span}\{\psi_1,\ldots,\psi_L\}$$

constitutes a low-rank approximation of $\mathscr{M}_{\Xi_N}$, optimal in some later defined sense.

To this end, we work at the algebraic level. Let $\mathbf{u}_h(\boldsymbol{\mu}^{(n)}) \in \mathbb{R}^M$ be the vector collecting the degrees of freedom (with respect to the FE basis) for the n-th snapshots $u_h(\boldsymbol{\mu}^{(n)})$, $n = 1,\ldots,N$, and consider the *snapshot matrix* $\mathbb{S} \in \mathbb{R}^{M\times N}$ storing such vectors in a column-wise sense, i.e.,

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

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

$$\mathbb{S} = \mathbb{W}\begin{bmatrix}\mathbb{D} & 0\\ 0 & 0\end{bmatrix}\mathbb{Z}^T = \mathbb{W}\,\Sigma\,\mathbb{Z}^T, \tag{5.8}$$

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

$$\mathbb{S}\mathbb{S}^T\mathbf{w}_m = \begin{cases}\sigma_m^2\,\mathbf{w}_m & \text{for } 1 \leq m \leq R,\\ \mathbf{0} & \text{for } R+1 \leq m \leq M,\end{cases} \tag{5.9}$$

$$\mathbb{S}^T\mathbb{S}\,\mathbf{z}_n = \begin{cases}\sigma_n^2\,\mathbf{z}_n & \text{for } 1 \leq n \leq R,\\ \mathbf{0} & \text{for } R+1 \leq n \leq N,\end{cases} \tag{5.10}$$

$$\mathbb{S}\,\mathbf{z}_i = \sigma_i\,\mathbf{w}_i \qquad \text{for } 1 \le i \le R, \tag{5.11}$$
$$\mathbb{S}^T\mathbf{w}_i = \sigma_i\,\mathbf{z}_i \qquad \text{for } 1 \le i \le R. \tag{5.12}$$

Due to the sparsity pattern of Σ in (5.8), the SVD of $\mathbb{S}$ can be cast in the compact form:

$$\mathbb{S} = \mathbb{W}_R\,\mathbb{D}\,\mathbb{Z}_R^T,$$

with $\mathbb{W}_R \in \mathbb{R}^{M\times R}$ and $\mathbb{Z}_R \in \mathbb{R}^{N\times R}$ retaining only the first R columns of W and Z, respectively, i.e.,

$$\mathbb{W}_R = \left[\mathbf{w}_1 \,\middle|\, \dots \,\middle|\, \mathbf{w}_R\right] \quad \text{and} \quad \mathbb{Z}_R = \left[\mathbf{z}_1 \,\middle|\, \dots \,\middle|\, \mathbf{z}_R\right]. \tag{5.13}$$

Exploiting the orthonormality of the columns of $\mathbb{W}_R$, the generic column $\mathbf{s}_n = \mathbf{u}_h\big(\boldsymbol{\mu}^{(n)}\big)$ of $\mathbb{S}$, $n = 1, \dots N$, can be expressed as [60]

$$\mathbf{s}_n = \sum_{r=1}^{R} (\mathbf{s}_n, \mathbf{w}_r)_{\mathbb{R}^M}\mathbf{w}_r\,,$$

where $(\cdot,\cdot)_{\mathbb{R}^M}$ denotes the Euclidean scalar product in $\mathbb{R}^M$. Therefore, the columns $\{\mathbf{w}_1, \dots, \mathbf{w}_R\}$ of $\mathbb{W}_R$ constitute an orthonormal basis for the column space Col($\mathbb{S}$) of $\mathbb{S}$.

Assume now we seek 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 span$\{\widetilde{\mathbf{w}}_1, \dots, \widetilde{\mathbf{w}}_L\}$ closest to $\mathbf{s}_n$ in the Euclidean norm $\|\cdot\|_{\mathbb{R}^M}$ is given by

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

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

$$\varepsilon(\widetilde{\mathbf{w}}_1, \dots, \widetilde{\mathbf{w}}_L) = \sum_{n=1}^{N} \left\| \mathbf{s}_n - \sum_{l=1}^{L} \big(\mathbf{s}_n, \widetilde{\mathbf{w}}_l\big)_{\mathbb{R}^M}\widetilde{\mathbf{w}}_l \right\|_{\mathbb{R}^M}^2 . \tag{5.14}$$

The Schmidt-Eckart-Young theorem [18, 58] 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 (5.14) 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

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

Hence, in the reduced basis problem (5.2) the test and trial functions are picked from the subspace $V_{\mathtt{rb}}$ of V_h spanned by the functions $\{\psi_l\}_{l=1}^L$, given by

$$\psi_l(\boldsymbol{x}) = \sum_{m=1}^{M} \psi_l^{(m)}\,\phi_m(\boldsymbol{x}) = \sum_{m=1}^{M} (\mathbf{w}_l)_m\,\phi_m(\boldsymbol{x}) \quad \text{for } 1 \le l \le L. \tag{5.15}$$

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

(a) if $M \le N$: directly solve the eigenvalue problems

$$\mathbb{S}\mathbb{S}^T\mathbf{w}_l = \lambda_l\,\mathbf{w}_l \quad \text{for } 1 \le l \le L;$$

(b) if $M > N$: compute the *correlation* matrix $\mathbb{M} = \mathbb{S}^T\mathbb{S}$ and solve the eigenvalue problems

$$\mathbb{M}\,\mathbf{z}_l = \lambda_l\,\mathbf{z}_l \quad \text{for } 1 \le l \le L.$$

Then, by (5.11) we have

$$\mathbf{w}_l = \frac{1}{\sqrt{\lambda_l}}\mathbb{S}\,\mathbf{z}_l \quad \text{for } 1 \le l \le L.$$

Algorithm 5.1 The offline and online stages for the POD-Galerkin (POD-G) RB method.

```
1: function $\mathbb{V}$ = PODG_OFFLINE($\mathscr{P}$, $\Omega_h$, $N$)
2:     generate the parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$
3:     compute the high-fidelity solutions $\{\mathbf{u}_h(\boldsymbol{\mu}^{(1)}), \dots, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})\}$ via FE-Newton's method
4:     generate the POD basis functions $\{\mathbf{w}_1, \dots, \mathbf{w}_L\}$ via method of snapshots
5:     assemble $\mathbb{V} = [\mathbf{w}_1 | \dots | \mathbf{w}_L]$
6: end function

1: function $\mathbf{u}_L(\boldsymbol{\mu})$ = PODG_ONLINE($\boldsymbol{\mu}$, $\mathbb{V}$)
2:     assemble and solve the reduced system (5.5) via Newton's method, yielding $\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu})$
3:     $\mathbf{u}_L(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu})$
4: end function
```

5.2. *Implementation: details and issues*

The numerical procedure presented so far can be efficiently carried out within an offline-online framework [51]. The parameter-independent *offline* step consists of the generation of the snapshots through a high-fidelity, expensive discretization 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* [53]. Then, given a new parameter value $\boldsymbol{\mu} \in \mathscr{P}$, the nonlinear reduced system (5.5) is solved *online* using, e.g., Newton's method, which entails the assembly and resolution of linear systems of the form (5.7). The main steps of the resulting POD-Galerkin (POD-G) RB method are summarized in Algorithm 5.1.

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. At this regard, notice that the operative definitions (5.5) and (5.6) of the reduced residual vector $\mathbf{G}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$ and its Jacobian $\mathbb{J}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$ involve the evaluation of the (high-fidelity) residual vector $\mathbf{G}_h(\cdot;\boldsymbol{\mu})$ and its Jacobian $\mathbb{J}_h(\cdot;\boldsymbol{\mu})$, whose cost clearly dependens on M. Moreover, due to the nonlinearity of the underlying PDE and the non-affinity in the parameter dependence (partially induced by the transformation map $\boldsymbol{\Phi}(\cdot;\boldsymbol{\mu})$), the assembly of the reduced linear systems (5.7) 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 successfully overcome upon resorting to suitable techniques such as the discrete empirical interpolation method (DEIM) [10] or its matrix variant (MDEIM) [49], aiming at recovering an affine dependency on the parameter $\boldsymbol{\mu}$. On the other hand, we should point out that the implementation of such techniques is problem dependent and of an *intrusive* nature, as it requires to access and 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].

6. A non-intrusive RB method using neural networks

The scenario portrayed so far motivates the research for an alternative approach to tackle any online query within the reduced basis framework, hopefully skipping the assembly and resolution of the reduced system. To this end, let us remark that there exists a one-to-one correspondence between the reduced space $V_{\mathtt{rb}}$ and the column space Col($\mathbb{V}$) of $\mathbb{V}$. Indeed, letting $\{\phi_1, \dots, \phi_M\}$ be a basis for V_h and $\{\psi_1, \dots, \psi_L\}$ be the reduced basis, from Eq. (5.15) follows:

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

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

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

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

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

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

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

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

$$u_h^{\mathbb{V}}(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{j=1}^{M} \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\right)_j \phi_j(\boldsymbol{x}) = \sum_{i=1}^{L} \left(\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\right)_i \psi_i(\boldsymbol{x})\,. \tag{6.1}$$

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

$$\begin{aligned} \boldsymbol{\pi} \,:\, \mathscr{P} \subset \mathbb{R}^P &\to \mathbb{R}^L \\ \boldsymbol{\mu} &\mapsto \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\,, \end{aligned} \tag{6.2}$$

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

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

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 entirely 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 $\boldsymbol{\pi}$.

In the literature, different approaches for the *interpolation* of (6.2) have been developed, e.g., exploiting some geometrical considerations concerning the solution manifold $\mathscr{M}$ [1], or employing radial basis functions [12]. In this work we resort to artificial neural networks, in particular multi-layer perceptrons, for the *nonlinear regression* of the map $\boldsymbol{\pi}$, leading to the POD-NN RB method. As described in Section 2.3, 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

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

According to the notation and nomenclature introduced previously, for $i = 1,\dots,N_{tr}$, $\mathbf{p}_i = \boldsymbol{\mu}^{(i)} \in \mathbb{R}^P$ represents the *input pattern* and $\mathbf{t}_i = \mathbb{V}^T\mathbf{u}_h\big(\boldsymbol{\mu}^{(i)}\big) \in \mathbb{R}^L$ the associated *teaching input*; together, they constitute a *training pattern*. In this respect, note that the teaching inputs $\mathbb{V}^T\mathbf{u}_h\big(\boldsymbol{\mu}^{(i)}\big)$, $i = 1,\dots,N_{tr}$, are generated through the FE solver. On the one hand, this ensures the reliability of the teaching patterns, given the assumed high-fidelity of the FE scheme On the other hand, this also suggests to incorporate the learning phase of the perceptron within the offline step of the POD-NN RB method, as described in Algorithm 6.1. In doing so, we exploit the natural decoupling between the training and the evaluation of neural networks, thus fulfilling the necessary requirement to enable great online efficiency.

However, 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 Section 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, resumed in the following.

While the Cybenko's theorems (see Section 2.2) allows 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 (to ease the work, we uniquely consider networks with the same number of neurons in both the first and the second hidden layer). 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 with respect to the number of training patterns). 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 we then proceed to re-train the networks. At this point, it is worth pointing out two remarks.

(i) Once the training set has been enriched, we can limit ourselves to network configurations including a number of neurons *non-smaller* than the current optimal network does. Indeed, the error (on the test data set) yielded by a neural network is decreasing in the number of patterns it 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 euristhic approach, employing, at each iteration, the latin hypercube sampling [33], which has proved to provide a good compromise between randomness and even covarage of the parameter domain; an example is offered in Fig. 6.1 for $\mathscr{P} \subset \mathbb{R}^2$.

The procedure is then iterated until a satisfactory degree of accuracy and generalization is attained, or the maximum number of training patterns is reached. Therefore, the speed up enabled by the pursuit of a neural network-based approach to tackle any *online* query comes at the cost of an extended *offline* phase.

In our numerical tests, we resort to the Levenberg-Marquardt algorithm to properly adjust the weights of the perceptron during the learning phase, relying on the mean squared error (MSE) (2.5) as performance function. To motivate this choice, let

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

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

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

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

$$\begin{aligned} MSE\big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}, \mathbb{V}^T\mathbf{u}_h\big) \propto \ & \big\|\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big\|_{\mathbb{R}^L}^2 = \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big)^T\big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big) \\ & = \big(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big)^T\big(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big) \\ & = \big\|\mathbf{u}_L^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big\|_{\mathbb{R}^M}^2 = \big\|u_L^{\mathtt{NN}} - u_h^{\mathbb{V}}\big\|_h^2 , \end{aligned} \tag{6.3}$$

where

$$u_L^{\mathtt{NN}}(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{i=1}^{L} \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})\big)_i \,\psi_i(\boldsymbol{x}) \in V_{\mathtt{rb}} . \tag{6.4}$$

Therefore, by minimizing the MSE, we actually 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 $V_{\mathtt{rb}}$ for all the training inputs $\boldsymbol{\mu}^{(i)}$, $i = 1, \dots, N_{tr}$. 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 [43], generalized cross validation [36]) aiming at preventing the network to *overfit* the training data.

In the following subsection, we aim at further investigating the accuracy which the proposed reduced basis strategy could provide. To this end, we develop a simplified *a priori* error analysis. Yet, no rigorous proof is provided; rather, the goal is to give some insights on the potential of the method.

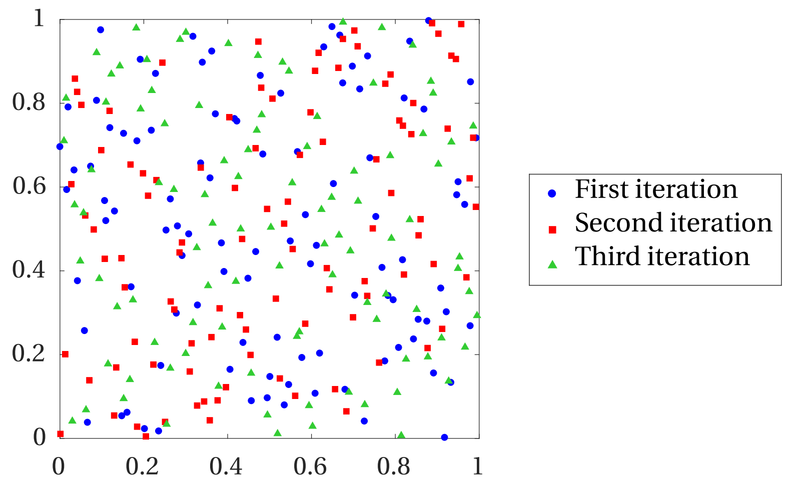

Figure 6.1. Three point sets randomly generate through the latin hypercube sampling. Observe the good coverage provided by the union of the sets, with only a few overlapping points.

Algorithm 6.1 The offline and online stages for the POD-NN RB method.

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

1: function $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu}) = \text{PODNN\_ONLINE}(\boldsymbol{\mu}, \mathbb{V}, \mathscr{N}_{\mathtt{rb}}, \mathscr{V}_{\mathtt{rb}}, \mathbf{w}_{\mathtt{rb}})$
2:     evaluate the output $\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$ of the network $(\mathscr{N}_{\mathtt{rb}}, \mathscr{V}_{\mathtt{rb}}, \mathbf{w}_{\mathtt{rb}})$ for the input vector $\boldsymbol{\mu}$
3:     $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$
4: end function
```

References

[1] Amsallem., D. (2010). *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.

[2] Ballarin, F., Manzoni, A., Quarteroni, A., Rozza, G. (2014). *Supremizer stabilization of POD-Galerkin approximation of parametrized Navier-Stokes equations.* MATHICSE Technical Report, École Polytechnique Fédérale de Lausanne.

[3] Barrault, M., Maday, Y., Nguyen, N. C., Patera, A. T. (2004). *An 'empirical interpolation' method: Application to efficient reduced-basis discretization of partial differential equations.* Comptes Rendus Mathematique, 339(9):667-672.

[4] Barthelmann, V., Novak, E., Ritter, K. (2000). *High-dimensional polynomial interpolation on sparse grids.* Advances in Computational Mathematics, 12(4):273-288.

[5] Bendsøe, M. P., Sigmund, O. (2004). *Topology optimization: Theory, methods and applications.* Heidelberg, DE: Springer Science & Business Media.

[6] Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., Mercer, R. L. (1993). *The mathematics of statistical machine translation: Parameter estimation.* Computational linguistics, 19(2):263-311.

[7] Buffa, A., Maday, Y., Patera, A. T., Prud'Homme, C., Turinici, G. (2012). *A priori convergence of the greedy algorithm for the parametrized reduced basis method.* ESAIM: Mathematical Modelling and Numerical Analysis, 46:595-603.

[8] Burkardt, J., Gunzburger, M., Lee, H. C. (2006). *POD and CVT-based reduced-order modeling of Navier-Stokes flows.* Computer Methods in Applied Mechanics and Egninnering, 196:337-355.

[9] Casenave, F., Ern, A., Lelièvre, T. (2015). *A nonintrusive reduced basis method applied to aeroacoustic simulations.* Advances in Computational Mathematics, 41:961-986.

[10] Chaturantabut, S., Sorensen, D. C. (2010). *Nonlinear model reduction via discrete empirical interpolation.* SIAM Journal on Scientific Computing, 32(5):2737-2764.

[11] Caloz, G., Rappaz, J. (1997). *Numerical analysis and bifurcation problems.* Handbook of numerical analysis, 5(2):487-637.

[12] Chen, W., Hesthaven, J. S., Junqiang, B., Yang, Z., Tihao, Y. (2017). *A greedy non-intrusive reduced order model for fluid dynamics.* Submitted to American Institute of Aeronautics and Astronautics.

[13] Cybenko, G. (1988). *Continuous valued neural networks with two hidden layers are sufficient.* Technical Report, Department of Computer Science, Tufts University.

[14] Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function.* Mathematics of Control, Signals, and Systems, 2(4):303–314.

[15] De Boor, C. (1978). *A practical guide to splines.* New York, NY: Springer-Verlag.

[16] Deparis, S. (2008). *Reduced basis error bound computation of parameter-dependent Navier-Stokes equations by the natural norm approach.* SIAM Journal of Numerical Analysis, 46(4):2039-2067.

[17] Dhondt, G. (2014). *CalculiX CrunchiX user's manual.* Available at `http://web.mit.edu/calculix_v2.7/CalculiX/ccx_2.7/doc/ccx/node1.html`.

[18] Eckart, C., Young, G. (1936). *The approximation of one matrix by another of lower rank.* Psychometrika, 1:211-218.

[19] Eftang, J. L. (2008). *Reduced basis methods for partial differential equations.* Master thesis, Department of Mathematical Sciences, Norwegian University of Science and Technology.

[20] Elman, H. C., Silvester, D. J., Wathen, A. (2004). *Finite elements and fast iterative solvers with applications in incompressible fluid dynamics.* New York, NY: Oxford University Press.

[21] Fahlman, S. E. (1988). *An empirical study of learning speed in back-propagation networks.* Technical Report CMU-CS-88-162, CMU.

[22] Gill, P. E., Murray, W., Wright, M. H. (1981). *Practical optimization.* Academic Press.

[23] Hagan, M. T., Menhaj, M. B. (1994). *Training feedforward networks with the Marquardt algorithm.* IEEE Transactions on Neural Networks, 5(6):989-993.

[24] Hagan, M. T., Demuth, H. B., Beale, M. H., De Jesús, O. (2014). *Neural Network Design, 2nd Edition.* Retrieved from `http://hagan.okstate.edu/NNDesign.pdf`.

[25] Hassdonk, B. (2013). *Model reduction for parametric and nonlinear problems via reduced basis and kernel methods.* CEES Computational Geoscience Seminar, Stanford University.

[26] Haykin, S. (2004). *Neural Networks: A comprehensive foundation.* Upper Saddle River, NJ: Prentice Hall.

[27] Hebb, D. O. (1949). *The organization of behaviour: A neuropsychological theory.* New York, NY: John Wiley & Sons.

[28] Liang, Y. C., Lee, H. P., Lim, S. P., Lin, W. Z., Lee, K. H., Wu, C. G. (2002) *Proper Orthogonal Decomposition and its applications - Part I: Theory.* Journal of Sound and Vibration, 252(3):527-544.

[29] Hassibi, B., Stork, D. G. (1993). *Second order derivatives for network pruning: Optimal Brain Surgeon.* Advances in neural information processing systems, 164-171.

[30] Hesthaven, J. S., Stamn, B., Rozza, G. (2016). *Certified reduced basis methods for parametrized partial differential equations.* New York, NY: Springer.

[31] Hesthaven, J. S., Stamn, B., Zhang, S. (2014). *Efficienty greedy algorithms for high-dimensional parameter spaces with applications to empirical interpolation and reduced basis methods.* ESAIM: Mathematical Modelling and Numerical Analysis, 48(1):259-283.

[32] Hopfield, J. J. (1982). *Neural networks and physical systems with emergent collective computational abilities.* Proceedings of the National Acadamedy Science, 79:2554-2558.

[33] Imam, R. L. (2008). *Latin hypercube sampling.* Encyclopedia of Quantitative Risk Analysis and Assessment.

[34] Jaggli, C., Iapichino, L., Rozza, G. (2014). *An improvement on geometrical parametrizations by transfinite maps.* Comptes Rendus de l'Académie des Sciences Paris, Series I, 352:263-268.

[35] Kaelbling, L. P., Littman, M. L., Moore, A. W. (1996). *Reinforcement Learning: A Survey.* Journal of Artificial Intelligence Reserch, 4:237-285.

[36] Kohavi, R. (1995). *A study of cross-validation and bootstrap for accuracy estimation and model selection.* Proceedings of the 40^{th} International Joint Conference on Artificial Intelligence, 2(12):1137-1143.

[37] Kohonen, T. (1998). *The self-organizing map.* Neurocomputing, 21(1-3):1-6.

[38] Kriesel, D. (2007). *A Brief Introduction to Neural Networks.* Retrieved from `http://www.dkriesel.com/en/science/neural_networks`.

[39] Le Maître, O., Knio, O. M. (2010). *Spectral methods for uncertainty quantification with applications to computational fluid dynamics.* Berlin, DE: Springer Science & Business Media.

[40] Lee, E. B., Markus, L. (1967). *Foundations of optimal control theory.* New York, NY: John Wiley & Sons.

[41] Maday, Y. (2006) *Reduced basis method for the rapid and reliable solution of partial differential equations.* Proceedings of the International Congress of Mathematicians, Madrid, Spain, 1255-1269.

[42] Marquardt, D. W. (1963). *An algorithm for least-squares estimation of nonlinear parameters*. Journal of the Society for Industrial and Applied Mathematics, 11(2):431-441.

[43] The MathWorks, Inc. (2016). *Machine learning challenges: Choosing the best model and avoiding overfitting*. Retrieved from `https://it.mathworks.com/campaigns/products/offer/common-machine-learning-challenges.html`.

[44] Mitchell, W., McClain, M. A. (2010). *A collection of 2D elliptic problems for testing adaptive algorithms*. NISTIR 7668.

[45] Manzoni, A., Negri, F. (2016). *Automatic reduction of PDEs defined on domains with variable shape*. MATHICSE technical report, École Polytechnique Fédérale de Lausanne.

[46] Møller, M. D. (1993). *A scaled conjugate gradient algorithm for fast supervised learning*. Neural Networks, 6:525-533.

[47] McClelland, J. L., Rumelhart, D. E. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, UK: MIT Press.

[48] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

[49] Negri, F., Manzoni, A., Amsallem, D. (2015). *Efficient model reduction of parametrized systems by matrix discrete empirical interpolation*. Journal of Computational Physics, 303:431-454.

[50] Persson, P. O. (2002). *Implementation of finite-element based Navier-Stokes solver*. Massachussets Institue of Technology.

[51] Prud'homme, C., Rovas, D. V., Veroy, K., Machiels, L., Maday, Y., Patera, A. T., Turinici, G. (2002). *Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods*. Journal of Fluids Engineering, 124(1):70-80.

[52] Quarteroni, A. (2010). *Numerical models for differential problems* (Vol. 2). New York, NY: Springer Science & Business Media.

[53] Quarteroni, A., Manzoni, A., Negri, F. (2015). *Reduced basis methods for partial differential equations: An introduction* (Vol. 92). New York, NY: Springer, 2015.

[54] Rannacher, R. (1999). *Finite element methods for the incompressible Navier-Stokes equations*. Lecture notes, Institute of Applied Mathematics, University of Heidelberg.

[55] Riedmiller, M., Braun, H. (1993). *A direct adaptive method for faster backpropagation learning: The rprop algorithm*. Neural Networks, IEEE International Conference on, 596-591.

[56] Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 65:386-408.

[57] Rudin, W. (1964). *Principles of mathematical analysis* (Vol. 3). New York, NY: McGraw-Hill.

[58] Schmidt, E. (1907). *Zur theorie der linearen und nichtlinearen integralgleichungen. I. Teil: Entwicklung willkürlicher funktionen nach systemen vorgeschriebener*. Mathematische Annalen, 63:433-476.

[59] Stergiou, C., Siganos, D. (2013). *Neural Networks*. Retrieved from `https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introductiontoneuralnetworks`.

[60] Volkwein, S. (2008). *Model reduction using proper orthogonal decomposition*. Lecture notes, Department of Mathematics, University of Konstanz.

[61] Widrow, B., Hoff, M. E. (1960). *Adaptive switching circuits*. Proceedings WESCON, 96-104.