POLITECNICO DI MILANO
SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
DEPARTMENT OF MATHEMATICS

Reduced order modeling of nonlinear problems using neural networks

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

in

Mathematical Engineering

Supervisors
Prof. Paolo Zunino
Prof. Jan S. Hesthaven

Candidate
Stefano Ubbiali
Personal Id. Number 836697

Academic Year 2016 – 2017

Abstract

In this thesis, 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.

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

Sommario

In questa tesi, si propone un metodo a basi ridotte (*reduced basis*, RB in inglese) non intrusivo per equazioni alle derivate parziali stazionarie parametriche. Il metodo estrae una base ridotta da un insieme di soluzioni altamente accurate tramite decomposizione ortogonale propria (*proper orthogonal decomposition*, POD) ed utilizza una rete neurale artificiale (*artificial neural network*, ANN), in particolare un percettrone multistrato (*multi-layer perceptron*, MLP), per approssimare accuratamente i coefficienti del modello ridotto. La ricerca del numero ottimale di neuroni per strato nascosto e del numero minimo di esempi di apprendimento necessario per non incorrere in *overfitting*, viene portata a termine nella fase *offline* tramite una routine automatica, combinando un campionamento a ipercubi latini (*Latin Hypercube Sampling*, LHS) con l'algoritmo di apprendimento di Levenberg-Marquardt. Questo garantisce un totale disaccoppiamento tra la fase *offline* e quella *online* che porta ad un metodo a basi ridotte efficiente - denominato POD-NN - adatto anche per problemi non lineari con una dipendenza non affine dai parametri. Si presentano studi numerici per l'equazione di Poisson (sia lineare che non lineare) e per fluidi viscosi in cavità, modellati tramite le equazioni di Navier-Stokes stazionarie ed incomprimibili. Diversi risultati confermano l'accuratezza del metodo POD-NN e mostrano il sostanziale vantaggio computazionale offerto nella fase *online* rispetto ad un metodo a basi ridotte tradizionale, basato su un processo di proiezione alla Galerkin.

Parole chiavi: metodi a basi ridotte non intrusivi, decomposizione ortogonale propria, percettrone multistrato, algoritmo di Levenberg-Marquardt, equazione di Poisson, fluido in cavità.

Contents

List of Figures

List of Algorithms

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 $\boldsymbol{\mu}$ [18, 29, 33]. A list of notable examples includes parameter estimation [6], topology optimization [5], optimal control [39] and uncertainty quantification [38]. 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 [52]. 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 [40]. 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 decomposition (POD) [58] and greedy algorithms [30]. 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 involving the computation of many fewer high-fidelity solutions [29]. 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 [52]. 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, 48].

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 informations [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 project, 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.

As their biological counterpart, *artificial neural networks* (ANNs) [25] are computing systems consisting of information-processing units, called *neurons*, interconnected through *weighted synapses*. The attractive feature of ANNs lies in their capability of *learning* from experience [37]. As a result, ANNs have found most used in those applications, such as cluster detection, image identification or speech recognition, which are difficult to address via computer algorithms using rule-based programming [42]. However, ANNs have been successfully applied also in more traditional contexts, e.g., continuous function approximation, as a valid alternative to the programming approach to investigate complexity, nonlinearity and uncertainties of a high order [47].

The learning of an ANN is accomplished through a *training* process, during which the network is exposed to a collection of examples and its weights are properly adjusted so that it can provide reasonable responses in similar (yet not identical) situations. As we will see, the training and the subsequent evaluation of a neural network perfectly fit the offline-online framework, offering the POD-NN method a significant online speed-up with respect to the standard projection-based POD-Garlerkin (POD-G) RB procedure.

Outline of the thesis

The dissertation is organized as follows. In Chapter 1, we offer an overview on (artificial) neural networks. In detail, we focus on the similarities with the human nervous system (Section 1.1), the mathematical modeling of a neuron (Section 1.2.1) and how neurons can be interconnected in a network (Section 1.2.2), and the backpropagation training algorithm for multi-layer perceptrons (Section 1.2.3).

The theoretical rational behind the POD-NN RB method is provided in Section 2.6 of Chapter 2. Preliminary to this, Sections 2.1, 2.2 and 2.3 rigorously define the functional framework, Section 2.4 introduces the finite element method and Section 2.5 details the standard POD-Galerkin RB strategy.

Numerical studies concerning the POD-G and the POD-NN methods are presented in Chapter 3 for linear and nonlinear diffusion problems in one (Section 3.1) and two (Section 3.2) spatial dimension, and for two-dimensional driven cavity flows (Section 3.3). The one-dimensional test cases considered in this work involve a physical parametrization, whereas the two-dimensional benchmarks involve only geometrical parameters. In Section 3.2.3, we also provide a comparison (in terms of accuracy and performance) with a traditional interpolation technique, based on Chebyshev points and cubic splines.

Lastly, we draw some concluding considerations and suggest possible future developments.

Chapter 1

Biological and artificial neural networks

Let us start this dissertation by introducing the key components of an artificial neural network and discussing the way it can be configured for a specific application. Please note that this chapter is not meant to provide a comprehensive overview on neural networks, rather to investigate some aspects and concepts functional to the following chapters. For further reading, we refer the reader to, e.g., [23, 25, 37], from which we retrieved many of the informations provided in this chapter.

Throughout this work we confine the attention to the most popular neural network paradigm - the *feedforward* neural network, presented in Section 1.2.2 - employing the well-known *backpropagation of error* as learning rule, derived in Section 1.2.3. In our numerical experiments, whose results will be discussed in Chapter 3, we mainly resort to a variant of backpropagation: the Levenberg-Marquardt algorithm.

Before moving to the description of technical neural networks, let us provide a brief introduction to their biological counterparts. The goal is to highlight the basic features of the human nervous system, focusing on the working principles of neurons and the way informations are processed, thus to extract the key concepts which should be taken into account for a mathematical representation.

1.1 Biological motivation

The information processing system of a vertebrate can coarsely be divided into the *central nervous system* (CNS) and the *peripheral nervous system* (PNS). The former consists of the *brain* and the *spinal cord*, while the latter mainly comprises the *nerves*, which transmit informations from all other parts of the body to the CNS (*sensory nerves*) and viceversa (*motor nerves*). When an output stimulus hits the sensory cells of an organ sense, these generate an electric signal, called *sensory signal*, which is transfered to the central nervous system via the *sensory nerves*. Within the CNS, informations are stored and managed to provide the muscles with a suitable *motor signal*, broadcast through the *motor nerves* and finally converted by the effectors into a system output [25].

Hence, both the central and peripheral nervous system are directly involved in the information processing workflow. At the cellular level, this is accomplished through a huge amount of modified cells called *neurons*. These processing elements continuosly communicate with each other by means of electric signals, traveling through a thick net

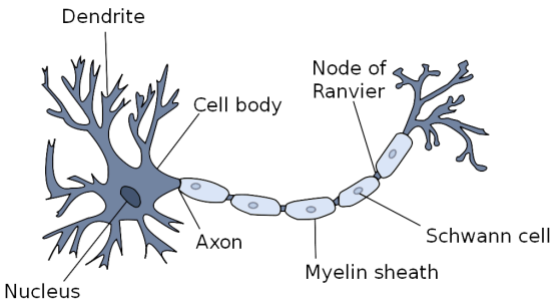


Figure 1.1. Simplified representation of a biological neuron, with the components discussed in the text; retrieved from [37].

of connections. For instance, in a human being each neuron is linked with $10^3 - 10^4$ other neurons in average. As detailed in the next paragraph, a neuron is characterized by a rather simple structure, specifically designed to rapidly collect input signals and generate an output pulse whenever the accumulated incoming signal exceeds a threshold: the *action potential*. In other terms, a neuron acts as a switch, establishing a typically nonlinear input-output mapping [37].

From a simplifyed perspective, a neuron consists of three main components: the *dendrites*, the *nucleus* or *soma*, and the *axon*. Dendrites are tree-like networks of nerve fibers receiving input signals from many sources and conveying them directly to the nucleus of the neuron. Here, input signals are accumulated and thresholded, as mentioned before. The possible output pulse is then broadcast to the cells contiguous the neuron through the axon - a unique, slender fiber constituing an extension of the soma and splitting in many branches at the opposite extremity [57]. To ease the electrical conduction of the signal, the axon is isolated through a myelin sheath which consists of Schwann cells (in the PNS) or oligodendrocytes (in the CNS). This insulating film is not continuous, rather presents gaps at regular intervals called *nodes of Ranvier*, which allow the signal to be conducted in a saltatory way.

The signal coming from the axon of another neuron or from another cell is transfered to the dendrites of a neuron through a particular connection called *synapsis* [1]. A synaptic may be either electrical or chemical. In the former, the presynaptic side, i.e., the sender axon, and the postsynapic side, i.e., the receiver dendrite, are directly in contact, so that the potential can simply travel by electrical conduction. Conversely, a chemical synapsis consists of a synaptic *cleft*, physically separating the presynaptic side from the postsynaptic side. Then, to let the action potential reach the postsynaptic side, at the presynaptic side the electrical pulse is converted into a chemical signal. This is accomplished by releasing some chemical substances called *neurotransmitters*. These neurotransmitters then cross the cleft and bind to the receptors dislocated onto the membrane of the postsynaptic side, where the chemical signal is re-converted into an electrical potential. However, neurotransmitters do not simply broadcast the action potential. Indeed, we can distinguish between excitatory and inhibitory neurotransmitters, respectively amplifying or modulating the signal. Hence,

[1] For the sake of completeness, we mention that there exist synapses directly connecting the axon of the sender neuron with either the soma or the axon of the receiver. Actually, a synapsis may also connect the axon of a neuron with the dendrite or soma of the same neuron (autosynapsis). However, for our purposes we can confine the attention to the axon-dendrite synapsis.

the pulse outgoing a neuron is preprocessed within the synapsis before reaching the target cell. In other terms, a neuron receives as input many *weighted* signals, which should then be collected.

Different studies have unveiled the tight correlation between the synapses which the neurons establish among each other, and the tasks a neural network can address [23]. That is, the set of interneuron connection strengths represent the information storage, i.e., the knowledge, of a neural network [37]. Knowledge is acquired through a *learning* or *training* process, entailing adjustments at the synaptic level to adapt to environmental situations. The adjustments may not only involve the modification of existing synapses, but also the creation of new synaptic connections. Hence, the nervous system is a distributed memory machine whose evolutionary structure is shaped by experience.

As mentioned above, a biological neural network acquaints itself with problems of a specific class through a learning procedure. During the learning process, the network is exposed to a collection of situations, giving it the possiblity to derive a set of tools which will let it provide reasonable solutions in similar circumstances. In other terms, the cognitive system should be able to *generalize*. Furthermore, after a successful training, a neural network should show a discrete level of *fault tolerance* against external errors, e.g. noisy inputs. Notice that the nervous system is also naturally fault tolerant against *internal* errors. Indeed, in case a neuron or a (relatively small) group of neurons is damaged or dies, the other processing nodes would take care of its tasks, so that the overall cognitive capabilities would be only slightly affected [37].

1.2 Artificial neural networks

Inspired by the biological information processing system discussed so far, 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 [47]. 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 [57].

Formally, a neural network could be defined as follows [37].

Definition 1.1 (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.

1.2.1 Neuronal model

As its name may suggest, an artificial neuron represents a simplified model of a biological neuron, retaining its main features discussed in Section 1.1. To introduce the components of the model, let us consider the neuron j represented in Fig. 1.2. 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 artificial neural networks 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:

$$f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) = \sum_{k=1}^{m} w_{s_k,j}\; y_{s_k}. \tag{1.1}$$

The function (1.1) provides a simple, yet effective way of modeling the accumulation of different input electric signals within a biological neuron; this motivates its popularity.

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 $\theta_j \in \mathbb{R}$ [37]:

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

where we have employed the weighted sum (1.1) as propagation function. From a biological perspective, the threshold θ_j is the analogous of the action potential mentioned in Section 1.1. Mathematically, it represents the point where the absolute value of the derivative of the activation function $|f'_{act}|$ is maximum. Then, the activation function reacts particularly sensitive when the net input u_j hits the threshold value θ_j [37].
Furthermore, noting that θ_j is a parameter of the network, one may choose to adapt it through a training process, exactly as it can be done for the synaptic weights, as we shall see in Section 1.2.3. However, θ_j is currently included in the activation function, making its runtime access cumbersome. This is typically overcome by introducing a *bias neuron* in the network. A bias neuron is a continuously firing neuron, with constant output $y_b = 1$,

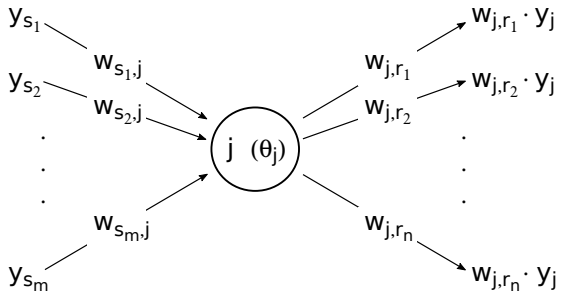


Figure 1.2. Visualization of the generic j-th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j}\ y_{s_1}, \dots, w_{s_m,j}\ y_{s_m}\}$ coming from the sending neurons $s_1, \dots, s_m$, and fires y_j, sent to the target neurons $\{r_1, \dots, r_n\}$ through the synapsis $\{w_{j,r_1}, \dots, w_{j,r_n}\}$. The neuron threshold θ_j is reported within its body.

which gets directly connected with neuron j, assigning the *bias weight* $w_{b,j} = -\theta_j$ to the connection. As can be deduced by Fig. 1.3, θ_j is now treated as a synaptic weight, while the neuron threshold is set to zero. Hence, the net input becomes

$$u_j = \sum_{k=1}^{m} w_{s_k,j}\ y_{s_k} - \theta_j\,, \tag{1.2}$$

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

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

Let us point out that this trick can be clearly applied to all neurons in the network which are characterized by a non-vanishing threshold: it suffices to connect the neuron with the bias, weighting the connection by the opposite of the threshold value. However, for ease of illustration, in the following we avoid to include the bias neuron in any graphical representation of a neural network.

Conversely to the propagation function, there exist various choices for the activation function, e.g., the Heaviside or binary function, which assumes only two values, 0 or 1, according to whether the argument is negative or positive, respectively:

$$f_{act}(v) = \begin{cases} 0\,, & \text{if } v < 0, \\ 1\,, & \text{if } v \geq 0. \end{cases} \tag{1.3}$$

Neurons characterized by such an activation function are usually named McCulloch-Pitts neurons, after the seminal work of McCulloch and Pitts [25]. In addition, note that (1.3) is discontinuous, with a vanishing derivative everywhere except in the origin, thus not admissible for the backpropagation training algorithm, as it will explained in Section 1.2.3.

Among continuous activation maps, sigmoid functions have been widely used for the realization of artificial neural networks due to their graceful combination of linear and nonlinear behaviour [25]. Sigmoid functions are s-shaped, monotically increasing, and assume values in a bounded interval, typically $[0, 1]$, as the logistic function,

$$f_{act}(v) = \frac{1}{1 + e^{-v/T}} \quad \text{with } T > 0\,, \tag{1.4}$$

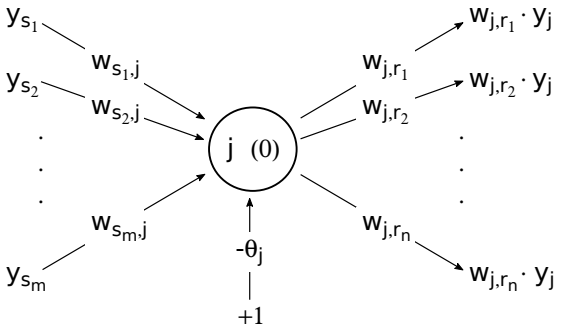


Figure 1.3. Visualization of the generic j-th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j}\ y_{s_1}, \ldots, w_{s_m,j}\ y_{s_m}, -\theta_j\}$ coming from the sending neurons $s_1, \ldots, s_m$, b, respectively, with b the bias neuron. The neuron output y_j is then conveyed towards the target neurons $\{r_1, \ldots, r_n\}$ through the synapsis $\{w_{j,r_1}, \ldots, w_{j,r_n}\}$. Observe that, conversely to the model offered in Fig. 1.2, the neuron threshold is now set to 0.

or $[-1,1]$, as the hyperbolic tangent,

$$f_{act}(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}. \tag{1.5}$$

Both functions are displayed in Fig. 1.4. Note that the logistic function resembles the Heaviside function as T decreases, i.e.,

$$\lim_{T\to 0} f_{act}(v) = 0 \quad \forall v < 0,$$
$$\lim_{T\to 0} f_{act}(v) = 1 \quad \forall v > 0.$$

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., $y_j = f_{out}(a_j) = a_j$. Let us point out that, while the input $\mathbf{p} = [y_{s_1}, \ldots, y_{s_m}]^T \in \mathbb{R}^m$ 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, including the outputting neuron itself (autosynapsis), 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 (see Section 1.2.2).
It is worth mentioning here that, as the activation function, also the output function is usually *globally* defined, i.e., all neurons in the network (or at least a group of neurons) are equipped with the same output function.

The neural model presented so far refers to the so called *computing* neuron, i.e., a neuron processing input informations 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) [25]. The role of source neurons will become clearer in the next section, where we will introduce the multi-layer feedforward neural network. Here, we just mention that such neuron receives a scalar unweighted input, which is simply forwarded to the connected neurons without performing any computation.

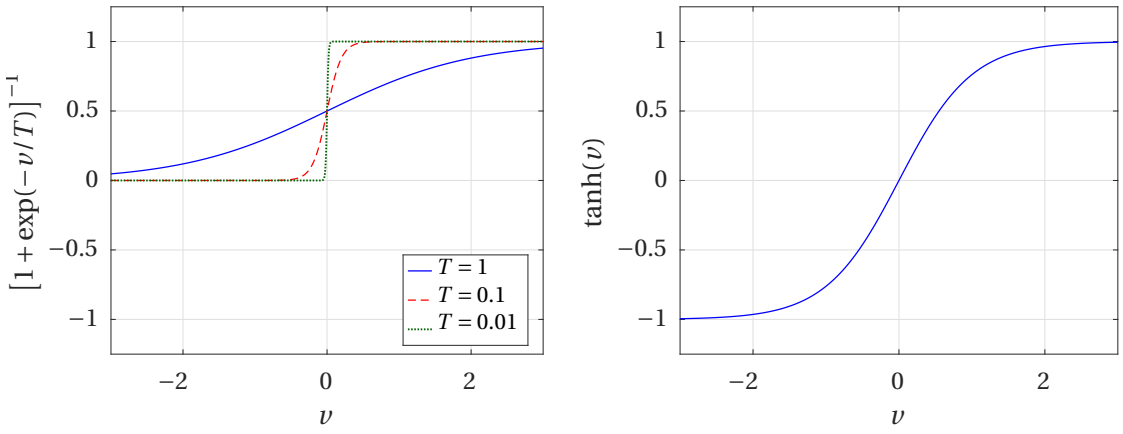


Figure 1.4. Left: logistic function (1.4) for three values of the parameter T; note that as T decreases, the logistic function resembles the Heaviside function. Right: hyperbolic tangent (1.5).

1.2.2 Network topologies: 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 or task. In this section, we expand our investigation for the two most common network topologies: the feedforward and the recurrent neural network.

Feedforward neural network. In a feedforward neural network, also called *perceptron* [55], 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, informations travel 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, response to the input pattern $\mathbf{p}$ [2]. 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, e.g., classification and 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. Note that the layer which the name SLP refers to is the output layer, while the input layer is not accounted for since no calculations, i.e., information processing operations, occur therein. Because of their quite simple structure, the range of application of SLPs is rather limited. For example, consider a

[2] 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., $q_j = y_j$, $j = 1, \dots, M_O$.

binary input vector and a unique output neuron, with a binary activation function. The SLP network acts then as a classifier, splitting the input space (i.e., the unit hypercube) by means of a hyperplane. Therefore, only *linearly separable* data can be properly represented using SLPs [37]. However, the share of linearly separable sets decreases as the space dimension increases, making SLPs seldom attractive.

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. Considering again the binary classifier discussed above, one can represent any convex polygon using one hidden layer and any arbitrary set by using two hidden layers; further increasing the number of hidden neurons would not improve the representation capability of the network. 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. 1.5. 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.

Finally, let us just mention that, although we have previously stated that in a feedforward neural network a synapses can only connect pairs of neurons in contiguous layers, recent years have seen the development of different variants. For instance, *shortcut connections* skip one or more layers, while *lateral connections* takes place between neurons within the same layer. However, throughout this work we shall be faithful to the original definition of the perceptron.

Recurrent neural network. In recurrent networks any neuron can bind with any other neuron, but autosynapses are forbidden, i.e., the output of a neuron cannot be input into the same neuron at the next time step. If each neuron is connected with all other neurons, then the network is said *completely linked*. As a consequence, one can not distinguish input neurons from output neurons. The input of the network is represented by the initial *network state*, which is the set of activation states for all neurons in the network. Similarly, the overall network output is given by the final network state. Hence, communication between a recurrent neural network and the surrounding environment takes place through the states of the neurons. Examples of recurrent networks are the Hopfield networks [31], inspired by the behaviour of electrically charged particles within a magnetic field, and the self-organizing maps by Kohonen [36], highly suitable for cluster detection.

As mentioned in the introductory chapter, in this work we refer to neural networks for the approximation of the unknown map linking the parameters of a parametrized partial differential equation with the coefficients of the corresponding reduced basis solution. To

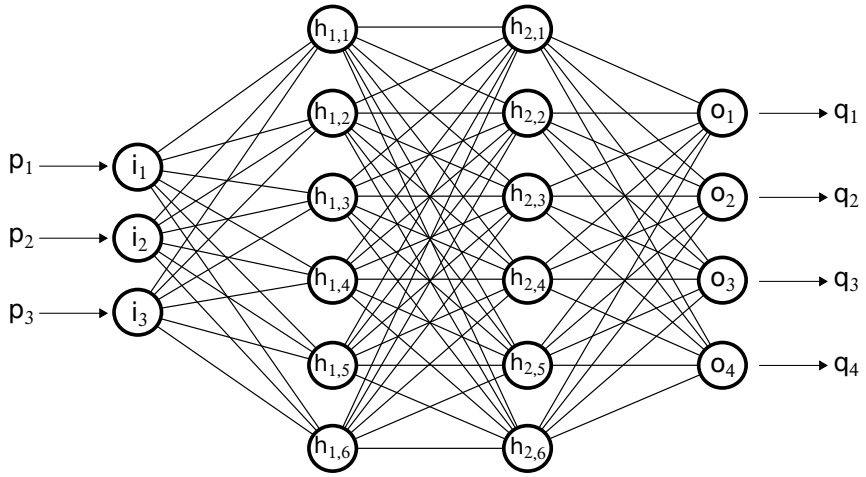


Figure 1.5. A three-layer feedforward neural network, with 3 input neurons, two hidden layers each one consisting of 6 neurons, and 4 output neurons. Within each connection, information flows from left to right.

accomplish this task, we rely on a collection of samples generated through a high-fidelity model. Although a detailed explanation will be provided in Chapter 2, it is worth noticing here that we are interested in a *continuous function approximation* problem. Therefore, a multi-layer feedforward neural network is the most suitable network architecture.

We will now investigate the way the weights of a perceptron can be *trained*.

1.2.3 Training a multi-layer feedforward neural network

As widely highlighted so far, the principal characteristic of a neural network is its capability of *learning* from the environment by storing the acquired knowledge through 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 neural network provides reasonable responses for unknown problems of the same class of the training set. This property is known as *generalization* [37].

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. The training set consists of a collection of *input patterns* (i.e., input vectors) $\{\mathbf{p}_i\}_{i=1}^{N_{tr}}$, and corresponding responses $\{\mathbf{t}_i\}_{i=1}^{N_{tr}}$, called *teaching inputs*. Then, $\mathbf{t}_i$ is the output the neural network should provide when it receives $\mathbf{p}_i$ as input. As we shall see, any training procedure aims to minimize (in some appropriate norm) the discrepancy between the *desired* output $\mathbf{t}_i$ and the *actual* output $\mathbf{q}_i$ given by the network as response to $\mathbf{p}_i$.

Unsupervised learning. Although supervised learning is a simple and intuitive paradigm, there exist many tasks which require a different approach, as, for instance, a cluster detection problem. Due to lack of prior knowledge, rather than telling the neural network how it

should behave in certain situations, one would like the neurons to *independently* identify rules to group items. Therefore, the training pattern is uniquely the input pattern. Since no associated output is provided, such a pattern is referred to as *unlabeled*, as opposed to the *labeled* examples involved in the supervised learning paradigm.

Reinforcement learning. Reinforcement learning is the most plausible paradigm from a biological viewpoint. After the completion of a series of input patterns, the neural network is supplied with a boolean or a real number telling whether the behaviour of the network is wrong or right. In the former case, the *feedback* or *reward* may also indicate to which extent the network is wrong [37]. Conversely to the supervised and unsupervised paradigms, reinforcement learning focuses on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Hence, this paradigm particularly suits problems involving a trade-off between a long-term and a short-term reward [34].

Clearly, the choice of the learning paradigm is task-dependent. In particular, for function approximation, i.e., what we are interested in, 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 $\{\mathbf{x}_i, \mathbf{y}_i = \boldsymbol{f}(\mathbf{x}_i)\}_{i=1}^{N_{tr}}$. The supervised learning paradigm aims at approximating $\boldsymbol{f}$ over a domain $V \subset \mathbb{R}^{M_I}$ up to a user-defined tolerance ϵ, i.e.,

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

where $\mathbf{F} : \mathbb{R}^{M_I} \to \mathbb{R}^{M_O}$ is the actual input-output map established by the neural network, and $||\cdot||$ is some suitable norm on $\mathbb{R}^{M_O}$. The system should provide accurate predictions for the input patterns, i.e.,

$$\boldsymbol{F}(\mathbf{x}_i) \approx \mathbf{y}_i, \quad \forall i = 1, \ldots, N_{tr}.$$

Then, the network is trained through a supervised learning algorithm, employing $\{\mathbf{x}_i\}_{i=1}^{N_{tr}}$ as input patterns and $\{\mathbf{y}_i\}_{i=1}^{N_{tr}}$ as teaching inputs; that is,

$$\mathbf{p}_i = \mathbf{x}_i \text{ and } \mathbf{t}_i = \mathbf{y}_i, \quad \forall i = 1, \ldots, N_{tr}.$$

As defined in the first part of this section, a training algorithm involves:

(a) a set of well-defined rules to modify the synaptic and bias weights;

(b) a *performance function*, quantifying the current level of knowledge of the surrounding environment.

Regarding (a), a plethora of weight updating techniques have been proposed in the literature, sometimes tailored to specific applications. Nevertheless, most of them rely on the well-known Hebbian rule, proposed by Donal O. Hebb in 1949 [26]. Inspired by neurobiological considerations, the rule can be stated in a two-steps fashion [25]:

(i) if two neurons on either side of a synapse (connection) are activated simultaneously, then the strength of that synapse is (selectively) increased;

(ii) if two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened (or eliminated).

Actually, (ii) was not included in the original statement; nevertheless, it provides a natural extension to [25]. In mathematical terms, consider the synapsis between a sending neuron i and a target neuron j. Then, 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 quantity

$$\Delta w_{i,j}(t) \sim \eta \; y_i(t) \; a_j(t)\,, \tag{1.6}$$

where $\eta > 0$ is the *learning rate*. We have also exploited the fact that $y_i = a_i$ since the output function is usually represented as the identity. Hence, at the subsequent iteration $t+1$ the synaptic weight is simply given by

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

Many of the supervised learning rules proposed in the literature, including the backpropagation of error, can be recast in the following form, which is a generalization of the Hebbian rule (1.6) [37]:

$$\Delta w_{i,j} = \eta \; h(y_i, w_{i,j}) \; g(a_j, t_j)\,, \tag{1.7}$$

where h and g depend on the specific learning rule and t_j is the j-th component of the teaching input $\mathbf{t}$. Note that all variables involved in (1.7) are supposed to be evaluated at time t, i.e., the correction $\Delta w_{i,j}$ is time-dependent. In addition, let us remark that (1.7) represents a *local* and *interactive* mechanism, since it involves both and only the neurons at the end-points of the synapse.

The second ingredient required to define a training algorithm is the performance or error function E. This function 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

$$E = E(\mathbf{w}) \in \mathbb{R}\,. \tag{1.8}$$

Recalling the notation and assumptions introduced in Definiton 1.1, here $\mathbf{w} \in \mathbb{R}^{|\mathcal{V}|}$ is a vector collecting the weights $\left\{w_{i,j} = w((i,j))\right\}_{(i,j)\in\mathcal{V}}$, with $\mathcal{V}$ the set of admissible connections in the network [3]. Thus, Eq. (1.8) implies that 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.

The steps for a generic supervised training procedure are listed by Algorithms 1.1 and 1.2 for online and offline learning, respectively. *Online* learning means that the weights are updated after the exposition of the network to each training pattern. In other terms,

[3] Please note that while in Definiton 1.1 $\mathcal{V}$ denoted the set of all *possible* connections, here $\mathcal{V}$ disregards those connections which are not consistent with the network topology in use. For instance, a feedforward neural network can not be endowed with connections oriented towards the input layer. Hence, such connections will not be included in $\mathcal{V}$. In this way, we reduce the size of $\mathbf{w}$ - which is a practical advantage.

Algorithm 1.1 Online supervised learning algorithm; the procedure ends when all training patterns yield an error below a defined threshold.

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 ϵ, maximum number of epochs T
Output: trained neural network $\left(\mathcal{N}, \mathcal{V}, \mathbf{w}_{opt}\right)$

$t = 0$, $i = 1$, $k = 0$
$\mathbf{w}(0) = \mathbf{w}_0$
while $t < T$ and $k < N_{tr}$ **do**
    evaluate output vector $\mathbf{y}_{\mathbf{p}_i}(t)$, corresponding to input pattern $\mathbf{p}_i$
    $E_{\mathbf{p}_i}(\mathbf{w}(t)) = d(\mathbf{y}_{\mathbf{p}_i}(t), \mathbf{t}_i)$
    if $E_{\mathbf{p}_i}(\mathbf{w}(t)) < \epsilon$ **then**
        $k \leftarrow k + 1$
    else
        $k = 0$
        compute weight update $\Delta\mathbf{w}(t)$ based on $E_{\mathbf{p}_i}(\mathbf{w}(t))$
        $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$
    end if
    $t \leftarrow t + 1$, $i \leftarrow i \pmod{N_{tr}} + 1$
end while
$\mathbf{w}_{opt} = \mathbf{w}(t)$

each epoch involves only one training pattern. Conversely, in an *offline* learning procedure the modifications are based on the entire training set, i.e., the weights are adjusted when the network has received all input patterns and the corresponding errors have been accumulated. Therefore, we should distinguish between the *specific error* $E_{\mathbf{p}}(\mathbf{w})$, specific to the activation pattern $\mathbf{p}$, and the *total error* $E(\mathbf{w})$ that accounts for all specific errors, namely

$$E(\mathbf{w}) = \sum_{\mathbf{p} \in P} E_{\mathbf{p}}(\mathbf{w}), \tag{1.9}$$

with P the training set. For instance, we could think of the specific error as the Mean Squared Error (MSE):

$$E_{\mathbf{p}}(\mathbf{w}) = \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\mathbf{p},j} - q_{\mathbf{p},j}\right)^2, \tag{1.10}$$

where the subscript $\mathbf{p}$ refers to the input pattern, $\mathbf{t}$ is the teaching input and $\mathbf{q}$ the actual output. Accordingly, the accumulated MSE is:

$$E(\mathbf{w}) = \sum_{\mathbf{p} \in P} E_{\mathbf{p}}(\mathbf{w}) = \sum_{\mathbf{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\mathbf{p}_i,j} - q_{\mathbf{p}_i,j}\right)^2. \tag{1.11}$$

So far, we have not discussed how the update $\Delta w_{i,j}$ for the weight $w_{i,j}$ is practically computed at each iteration. To this end, we have to rigourously define the functions h and g involved in the generalized Hebbian rule (1.7). Let us recall that any operation of the neural network can be seen as a point over the error surface $E(\mathbf{w})$. Therefore, to increase the performance of the network, we need to iteratively move towards a (local) minimum of the

Algorithm 1.2 Offline 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 ϵ, maximum number of epochs T
Output: trained neural network $(\mathcal{N}, \mathcal{V}, \mathbf{w}_{opt})$

$t = 0$
$\mathbf{w}(0) = \mathbf{w}_0$
$E(\mathbf{w}(0))$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}(0)$, P, d)
while $t < T$ and $E(\mathbf{w}(t)) > \epsilon$ **do**
    compute weight update $\Delta\mathbf{w}(t)$ based on $E(\mathbf{w}(t))$
    $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$
    $E(\mathbf{w}(t+1))$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}(t+1)$, P, d)
    $t \leftarrow t+1$
end while
$\mathbf{w}_{opt} = \mathbf{w}(t)$
function $E(\mathbf{w})$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}$, P, d)
    $E(\mathbf{w}) = 0$
    for $i = 1, \ldots, N_{tr}$ **do**
        evaluate output vector $\mathbf{y}_{\mathbf{p}_i}$, corresponding to input pattern $\mathbf{p}_i$
        $E(\mathbf{w}) \leftarrow E(\mathbf{w}) + d(\mathbf{y}_{\mathbf{p}_i}, \mathbf{t}_i)$
    end for
end function

surface [25]. For this purpose, we may employ a *steepest descent* technique, thus following the direction of the anti-gradient, i.e.,

$$\Delta w_{i,j} = -\eta \frac{\partial E(\mathbf{w})}{\partial w_{i,j}}, \tag{1.12}$$

where $\eta > 0$ is the learning rate that modulates the size of the step. Among the others, the *backpropagation of error* [46] is surely the most-known supervised, gradient-based training procedure for a multi-layer feedforward neural network whose neurons are equipped with a *semi-linear*, i.e., continuous and differentiable, activation function [4]. The derivation of the backpropagation algorithm is provided in the following.

Backpropagation of error

Let us consider the generic synapse (i, j) of a multi-layer feedforward neural network, connecting the *predecessor* neuron i with the *successor* neuron j. As mentioned before, suppose both i and j present a semi-linear activation function. By exploiting the chain rule, the backpropagation of error provides an operative formula for the evaluation of the antigradient of the error function $E(\mathbf{w})$ in an arbitrary point $\mathbf{w}$, thus allowing to compute the update $\Delta w_{i,j}$ for the weight $w_{i,j}$ according to (1.12). For this purpose, we need to distinguish whether the successor neuron j is either an output or an inner neuron. To

[4] Therefore, backpropagation cannot be applied with a binary activation function.

improve the clarity of the following mathematical derivation, we add the subscript $\mathbf{p}$ to any variable concerning a neuron.

Let $S = \{s_1, \dots, s_m\}$ be the set of m neurons sending their output to neuron j through the synapses $\{(s_1, j), \dots, (s_m, j)\}$. Recalling the definition (1.9) of the accumulated error, via the chain rule we can formulate the derivative of $E(\mathbf{w})$ with respect to the weight $w_{i,j}$ as follows:

$$\frac{\partial E(\mathbf{w})}{\partial w_{i,j}} = \sum_{\mathbf{p}\in P} \frac{\partial E_{\mathbf{p}}(\mathbf{w})}{\partial w_{i,j}} = \sum_{\mathbf{p}\in P} \frac{\partial E_{\mathbf{p}}(\mathbf{w})}{\partial u_{\mathbf{p},j}} \frac{\partial u_{\mathbf{p},j}}{\partial w_{i,j}}. \tag{1.13}$$

Assuming the propagation function be represented as the weighted sum, so that the net input is given by (1.2), the last factor in (1.13) can be written as

$$\frac{\partial u_{\mathbf{p},j}}{\partial w_{i,j}} = \frac{1}{\partial w_{i,j}} \left(\sum_{s\in S} w_{s,j}\ y_{\mathbf{p},s} - \theta_j \right) = y_{\mathbf{p},i}\,. \tag{1.14}$$

We then denote the opposite of the first term in (1.13) by $\delta_{\mathbf{p},j}$, i.e.,

$$\delta_{\mathbf{p},j} = -\frac{\partial E_{\mathbf{p}}(\mathbf{w})}{\partial u_{\mathbf{p},j}}. \tag{1.15}$$

$\delta_{\mathbf{p},j}$ is often referred to as the *sensitivity* of the specific error $E_{\mathbf{p}}$ with respect to neuron j. Inserting (1.15) and (1.14) into (1.13) and embedding the resulting equation into (1.12), the weight update can be written as:

$$\Delta w_{i,j} = \eta \sum_{\mathbf{p}\in P} \delta_{\mathbf{p},j}\ y_{\mathbf{p},i}\,. \tag{1.16}$$

We now proceed to derive an operative formula for $\delta_{\mathbf{p},j}$. When j is an output neuron, and using an identity output function, it follows that:

$$\begin{aligned}
\delta_{\mathbf{p},j} &= -\frac{\partial E_{\mathbf{p}}(\mathbf{w})}{\partial y_{\mathbf{p},j}} \frac{\partial y_{\mathbf{p},j}}{\partial u_{\mathbf{p},j}} \\
&= -\frac{1}{\partial y_{\mathbf{p},j}} \frac{1}{M_O} \sum_{k=1}^{M_O} \left(t_{\mathbf{p},k} - y_{\mathbf{p},k}\right)^2 \frac{\partial a_{\mathbf{p},j}}{\partial u_{\mathbf{p},j}} \\
&= \frac{2}{M_O} \left(t_{\mathbf{p},j} - y_{\mathbf{p},j}\right) \frac{\partial f_{act}(u_{\mathbf{p},j})}{\partial u_{\mathbf{p},j}} \\
&= \frac{2}{M_O} \left(t_{\mathbf{p},j} - y_{\mathbf{p},j}\right) f'_{act}(u_{\mathbf{p},j}),
\end{aligned} \tag{1.17}$$

where we have used the specific error given in (1.10). As appears in (1.17), we must require a differentiable transfer function.

It is worth mentioning that (1.17) does not apply when j lies within a hidden layer as no teaching input is provided for a hidden neuron. In that case, we denote by $R = \{r_1, \dots, r_n\}$ the set of n neurons receveing the output generated by j, called the *successors*. We then

note that the output of any neuron can directly affect only its successors [37]. Hence:

$$\begin{aligned}\delta_{\mathbf{p},j} &= \frac{\partial E_{\mathbf{p}}(\mathbf{w})}{\partial u_{\mathbf{p},j}} \\ &= \frac{\partial E_{\mathbf{p}}(u_{\mathbf{p},r_1},\ldots,u_{\mathbf{p},r_n})}{\partial u_{\mathbf{p},j}} \\ &= \frac{\partial E_{\mathbf{p}}(u_{\mathbf{p},r_1},\ldots,u_{\mathbf{p},r_n})}{\partial y_{\mathbf{p},j}}\,\frac{\partial y_{\mathbf{p},j}}{\partial u_{\mathbf{p},j}} \\ &= \sum_{k=1}^{n}\frac{\partial E_{\mathbf{p}}}{\partial u_{\mathbf{p},r_k}}\,\frac{\partial u_{\mathbf{p},r_k}}{\partial y_{\mathbf{p},j}}\,\frac{\partial y_{\mathbf{p},j}}{\partial u_{\mathbf{p},j}}\,.\end{aligned} \tag{1.18}$$

Applying the definition of sensitivity for neuron r_k, $k = 1,\ldots,n$, under the same assumptions of Eq. (1.17) we can further develop (1.18):

$$\begin{aligned}\delta_{\mathbf{p},j} &= \sum_{k=1}^{n}\left(-\delta_{\mathbf{p},r_k}\right)\,\frac{1}{\partial y_{\mathbf{p},j}}\left(\sum_{l=1}^{m_k} w_{s_l,r_k}\,y_{\mathbf{p},s_l} - \theta_{r_k}\right)\,\frac{\partial f_{act}(u_{\mathbf{p},j})}{\partial u_{\mathbf{p},j}} \\ &= -\sum_{k=1}^{n}\delta_{\mathbf{p},r_k}\;w_{j,r_k}\;f'_{act}(u_{\mathbf{p},j})\,.\end{aligned} \tag{1.19}$$

Here, we have supposed that r_k receives input signals from m_k neurons $\{s_1,\ldots s_{m_k}\}$.

Let us now summarize the results derived so far. At any iteration t of the backpropagation learning algorithm, the weight $w_{i,j}(t)$ of a generic connection (i,j) linking neuron i with neuron j is corrected by an additive quantity $\Delta w_{i,j}(t)$, i.e.,

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

By using the accumulated MSE (1.11) as performance function and understanding the dependence on time for the sake of clarity, the weight update $\Delta w_{i,j}$ reads:

$$\Delta w_{i,j} = \eta\sum_{\mathbf{p}\in P} y_{\mathbf{p},i}\;\delta_{\mathbf{p},j}\,,$$

where $\eta > 0$ and $\delta_{\mathbf{p},j}$ is given by

$$\delta_{\mathbf{p},j} = \begin{cases} f'_{act}(u_{\mathbf{p},j})\sum_{r\in R}\delta_{\mathbf{p},r}\;w_{j,r}\,, & \text{if } j \text{ inner neuron}\quad, \qquad (1.20a)\\ \dfrac{2}{M_O}f'_{act}(u_{\mathbf{p},j})\left(t_{\mathbf{p},j}-y_{\mathbf{p},j}\right), & \text{if } j \text{ output neuron}\quad. \qquad (1.20b)\end{cases}$$

Some relevant remarks about the overall algorithm should be made. First, observe that (1.20a) defines $\delta_{\mathbf{p},j}$ for a hidden node j by relying on the neurons in the following layer, whereas (1.20b) only involves variables concerning the neuron itself (namely, $u_{\mathbf{p},j}$ and $y_{\mathbf{p},j}$) and the exact output $t_{\mathbf{p},j}$. Therefore, the coupled equations (1.20a)-(1.20b) implicitly set the order in which the weights must be adjusted: starting from the output layer, update all the 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 [37, 59].

The weight updating procedure detailed above corresponds to the offline version of the backpropagation of error, since it involves the total error E. However, the online algorithm

readly comes from (1.16) by simply dropping the summation over the elements of the training set P.

Although intuitive and very promising, backpropagation of error suffers from all drawbacks that are peculiar to gradient-based techniques. For instance, we may get stuck in a local minimum, whose level is possibly far from the global minimum of the error surface E. Furthermore, since the step size dictated by the gradient method is given by the norm of the gradient itself, minima close to steepest gradients are likely to be missed due to a large step size. This motivates the introduction in (1.12) of the learning rate $\eta \in (0,1]$, which acts as a reducing factor and thus enables a better control on the descent. As suggested by Kriesel [37], in many applications reasonable values for η lies in the range $[0.01, 0.9]$. In particular, a time-dependent learning rate usually enables a more effective and more efficient training procedure. On the one hand, at the beginning of the process, when the network is far from the application goal, one often needs to span a large extent of the error landscape, to identify a region of interest. At that time, the learning rate should be large, i.e., close to 1, to speed up the exploration. On the other hand, as we approach the optimal configuration, we may want to progressively reduce the learning rate, thus the step size, to fine-tune the weights of the neural network.

Nevertheless, selecting an appropriate value for η remains more an art than a science. Furthermore, the derivative of sigmoidal activations functions (see Fig. 1.4) is close to zero far from the origin. As a consequence, it becomes very difficult to move neurons away from the limits of the activation, which could extremely extend the learning time [37]. Hence, different alternatives to backpropagation have been proposed in the literature, either by modifying the original algorithm (as, e.g., the resilient backpropagation [54] or the quickpropagation [20]), or pursuing a different numerical approach to the optimization problem. The latter class of algorithms includes the Levenberg-Marquardt algorithm, which we use extensively in our numerical tests.

Levenberg-Marquardt algorithm

While backpropagation is a steepest descent algorithm, the Levenberg-Marquardt algorithm [41] is an approximation to Newton's method [22]. As for backpropagation, the learning procedure is driven by the loss function $E = E(\mathbf{w})$, $\mathbf{w} \in \mathbb{R}^{|\mathcal{V}|}$. Applying Newton's method for the minimization of E, at each iteration the *search direction* $\Delta\mathbf{w}$ is found by solving the following linear system:

$$\nabla^2 E(\mathbf{w})\Delta\mathbf{w} = -\nabla E(\mathbf{w}), \tag{1.21}$$

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 now that the loss function is represented as the accumulated MSE,

$$E(\mathbf{w}) = \sum_{\mathbf{p}\in P} \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\mathbf{p},j} - y_{\mathbf{p},j}\right)^2 = \sum_{\mathbf{p}\in P} \frac{1}{M_O} \sum_{j=1}^{M_O} e_{\mathbf{p},j}(\mathbf{w})^2;$$

with $e_{\mathbf{p},j}$ the j-th component of the error vector $\mathbf{e_p} = \mathbf{t_p} - \mathbf{y_p}$ corresponding to the input pattern $\mathbf{p}$. Then, introducing the Jacobian $J_\mathbf{p}$ of the specific error vector $\mathbf{e_p}$ with respect to

w, i.e.,

$$J_{\mathbf{p}}(\mathbf{w}) = \begin{bmatrix} \frac{\partial e_{\mathbf{p},1}}{\partial w_1} & \frac{\partial e_{\mathbf{p},1}}{\partial w_2} & \cdots & \frac{\partial e_{\mathbf{p},1}}{\partial w_{|\mathcal{V}|}} \\ \frac{\partial e_{\mathbf{p},2}}{\partial w_1} & \frac{\partial e_{\mathbf{p},2}}{\partial w_2} & \cdots & \frac{\partial e_{\mathbf{p},2}}{\partial w_{|\mathcal{V}|}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{\mathbf{p},M_O}}{\partial w_1} & \frac{\partial e_{\mathbf{p},M_O}}{\partial w_2} & \cdots & \frac{\partial e_{\mathbf{p},M_O}}{\partial w_{|\mathcal{V}|}} \end{bmatrix} \in \mathbb{R}^{M_O \times |\mathcal{V}|} \tag{1.22}$$

simple computations yield:

$$\nabla E(\mathbf{w}) = \sum_{\mathbf{p}\in P} \frac{2}{M_0} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}} \in \mathbb{R}^{|\mathcal{V}|} \tag{1.23}$$

and

$$\nabla^2 E(\mathbf{w}) = \sum_{\mathbf{p}\in P} \frac{2}{M_O} \left[J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w}) + S(\mathbf{w}) \right] \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}, \tag{1.24}$$

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 (1.23) and (1.24) into (1.21) we get the linear system

$$\left[\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w}) \right] \Delta\mathbf{w} = -\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}}(\mathbf{w})\,. \tag{1.25}$$

The Levenberg-Marquardt modification to Newton's method reads [22, 41]:

$$\left[\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w}) + \mu I \right] \Delta\mathbf{w} = -\sum_{\mathbf{p}\in P} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}}(\mathbf{w})\,, \tag{1.26}$$

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 (1.25), while for $\mu \gg 1$ the search direction $\Delta\mathbf{w}$ approaches antigradient of E, i.e., we recover the backpropagation algorithm. Then, 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.

The Levenberg-Marquardt training algorithm proceeds as follows. At each epoch t of the training procedure, we solve the (potentially large) linear system (1.26). 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 [41].

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., $w_k = w_{i,j}$. Since

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

is related with the gradient of the performance function, we can follow the same steps performed to derive the backpropagation, namely:

$$\begin{aligned}\frac{\partial e_{\mathbf{p},h}}{\partial w_{i,j}} &= \frac{\partial e_{\mathbf{p},h}}{\partial u_{\mathbf{p},j}}\,\frac{\partial u_{\mathbf{p},j}}{\partial w_{i,j}}\\ &= \frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}}\,\frac{\partial y_{\mathbf{p},j}}{\partial u_{\mathbf{p},j}}\,\frac{\partial u_{\mathbf{p},j}}{\partial w_{i,j}}\\ &= \frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}}\,f'_{act}(u_{\mathbf{p},j})\,y_{\mathbf{p},i}\\ &= \delta_{\mathbf{p},h,j}\,y_{\mathbf{p},i}\,,\end{aligned} \tag{1.27}$$

with

$$\delta_{\mathbf{p},h,j} := -\frac{\partial e_{\mathbf{p},h}}{\partial u_{\mathbf{p},j}} = -\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}}\,f'_{act}(u_{\mathbf{p},j})\,. \tag{1.28}$$

For the computation of the derivative

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

occurring in (1.28), we assume that, within the set of neurons $\mathcal{N}$, items are ordered such that the output neurons come first, i.e.,

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

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, similarly to (1.19) 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

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

with

$$\delta_{\mathbf{p},h,j} = \begin{cases} f'_{act}(u_{\mathbf{p},j}) \sum_{r \in R} \delta_{\mathbf{p},h,r}\ w_{j,r}\,, & \text{if } j \text{ inner neuron}\ , \quad (1.29a) \\ f'_{act}(u_{\mathbf{p},j}) \delta^K_{jh}\,, & \text{if } j \text{ output neuron}\ , \quad (1.29b) \end{cases}$$

where δ^K_{jh} is the Kronecker delta. The steps to be performed at each iteration of the Levenberg-Marquardt algorithm are summarized in Algorithm 1.3.

Let us finally remark that a trial and error approach is still required to find satisfactory values for μ and β; as proposed in [41], a good starting point may be $\mu = 0.01$, with $\beta = 10$. Moreover, the dimension of the system (1.26) increases nonlinearly with the number of neurons in the network, making the Levenberg-Marquardt algorithm poorly efficient for large networks [22]. However, it is more efficient than backpropagation for networks with a few hundreads of connections, besides producing much more accurate results. We will further develop this topic in Chapter 3.

Algorithm 1.3 An iteration of the Levenberg-Marquardt training algorithm.

function $\left[\mathbf{w} + \Delta\mathbf{w},\, E(\mathbf{w} + \Delta\mathbf{w}),\, \mu\right]$ = LMITERATION($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}$, P, $E(\mathbf{w})$, d, μ)
  $\beta = 10$
  for $i = 1, \ldots, N_{tr}$ **do**
    evaluate output vector $\mathbf{y}_{\mathbf{p}_i}$, corresponding to input pattern $\mathbf{p}_i$
    $\mathbf{e}_{\mathbf{p}_i} = \mathbf{y}_{\mathbf{p}_i} - \mathbf{t}_{\mathbf{p}_i}$
    for $h = 1, \ldots, M_0$, $k = 1, \ldots, |\mathcal{V}|$ **do**
      compute $\left(J_{\mathbf{p}_i}\right)_{h,k}$ according to (1.27), (1.29a) and (1.29b)
    end for
  end for
  assemble and solve $\left[\sum_{i=1}^{N_{tr}} J^T_{\mathbf{p}_i} J_{\mathbf{p}_i} + \mu I\right] \Delta\mathbf{w} = -\sum_{i=1}^{N_{tr}} J^T_{\mathbf{p}_i} \mathbf{e}_{\mathbf{p}_i}$
  $E(\mathbf{w} + \Delta\mathbf{w})$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w} + \Delta\mathbf{w}$, P_{tr}, d)
  if $E(\mathbf{w} + \Delta\mathbf{w}) < E(\mathbf{w})$ **then**
    $\mu \leftarrow \mu / \beta$
  else
    $\mu \leftarrow \mu * \beta$
  end if
end function

1.2.4 Practical considerations on the design of artificial neural networks

We conclude this introductory chapter on neural networks by discussing the major concerns regarding their design and implementation, mainly focusing on MLPs. As we shall see, most of these issues are still open questions in many research fields, and as such they should be tackled pursuing a *trial-and-error* approach.

In Section 1.2.2, we reported two fundamental results ((i) and (ii)) promoting three-layers feedforward neural networks as universal function approximators. Yet, these statements do not provide any information regarding the number of neurons and training patterns required to approximate a function up to a desired level of uncertainty. In other terms, (i) and (ii) are not operative, and therefore one has to rely on emprirical considerations and greedy approaches in the design and training of a neural network.

Clearly, the larger the training set, the better, and one could perform a sensitivity analysis on the amount of teaching inputs required to get satisfactory predictions. However, in real-life applications training data often come from an external source of information, on which we do not have any influence, and as a result the number of available teaching patterns is fixed. A relevant example is provided by recommender systems, which seek to build a predictive model based on a user's past behaviour (e.g., the items he/she has purchased) and similar choices made by other users.

The critical point for any network paradigm, designed either for continuous regression, classification or cluster-detection, is the choice of the right number of neurons it should be equipped with. As a rule of thumb, the network should feature as *few* free parameters as possible but as many as *necessary* [37]. In this respect, recall that the computing and processing power of a network is determined by its neurons, while the weighted synapses represent the information storage. On the one hand, too few neurons (i.e., synapses) do not endow the network with the necessary representation capability, leading to poor results. On the other hand, an oversized network precisely aligns with the teaching inputs, but is likely to fail on patterns it has not been exposed to during the training. Indeed, being the degrees of freedom of the underlying problem fewer than the network paremeters, the latter can be tuned to lower the error function at will. In other words, once the training phase is over, the system has successfully *memorized* the training patterns but is not capable of *generalizing* what it has learnt to similar (yet different) situations. In this circumstance, we say that the network *overfits* the training data.

In the past, many techniques have been proposed to avoid overfitting. For instance, one could alter the data through additive (white) noise, thus preventing the network to perfectly fit the training set. Another well-known practice is *regularization*, which consists in correcting the error function by a regularizing term $L(\mathbf{w})$, namely

$$E(\mathbf{w}) + \lambda L(\mathbf{w}) .$$

Here, L is a functional increasing with the components of $\mathbf{w}$, thus penalizing large values for the weights, and so preventing the network from heavily relying on individual data points [42]. The positive coefficient λ tunes the level of regularization introduced in the model: for $\lambda = 0$, we recover the original model, while for $\lambda \gg 1$ we sacrifice the performance on the training dataset for the sake of a more flexible system. As typical in neural networks, a suitable value for λ has to be found empirically.

In our numerical experiments, overfitting is prevented upon resorting to *cross-validation* combined with an *early stopping* technique [35]. Data are split in three subsets: *training*, *validation*, and *testing* data sets, denoted respectively by P_{tr}, P_{va} and P_{te}. The former consists of data which are actually used to train the network. Specifically, in the Levenberg-Marquardt algorithm these data are used to build up the linear system (1.26) yielding the weights and biases update $\Delta\mathbf{w}$. Validation samples are used to monitor the error performed by the model *during* the training, but are not involved in the training itself. Finally, the testing data set is used to assess the performance of the system once the learning phase is over, and it is thus useful to compare different models, e.g., neural networks with different number of layers and/or neurons per layer [42]. For further details on the way such subsets have been generated for our test cases, we refer the reader to the upcoming chapters.

At the beginning of the learning stage, the error on both the training and the validation data set typically decreases. However, while the error yielded by the training samples should

continue to decrease as time advances (provided a well-chosen minimization technique), at a certain point the validation error may start increasing, meaning that the network is likely to overfit the data. Therefore, we *early stop* the procedure whenever the validation error increases for K_{ea} consecutive iterations. The optimal configuration $\mathbf{w}_{opt}$ is then the one yielding the minimum of the validation (and not the training) error curve.

Once in possession of an effective training procedure, one can pursue two different yet complementary strategies to determine a suitable number of neurons for a given application:

(a) moving from a (relative) small amount of neurons, progressively augment the size of the network until the error on the test data set starts increasing;

(b) consider a sufficiently large number of neurons, then possibly adopt a *pruning* strategy (e.g., the Optimal Brain Surgeon technique proposed by Hassibi & Stork [28]) to iteratively remove the *less relevant* connections, erasing a neuron whenever it gets isolated from the rest of the network.

As we shall see in Section 2.6, in this work we resort to the first approach for the sake of implementation convenience. In particular, we first consider perceptrons with a single hidden layer, and we then add another computing layer whenever necessary [5].

Lastly, let us point out that the final network configuration resulting from any learning strategy is affected by the initial values assigned to the synaptic and bias weights. As good practice, the weights should be initialized with random values averaging zero. In this respect, a random uniform sampling over $[-0.5, 0.5]$ or a standard Gaussian distribution may be adequate choices. Then, to limit the dependence of the results on the initial configuration $\mathbf{w}_0$, we train each network topology several times, say K_{mr}, employing different $\mathbf{w}_0^k$, $k = 1, \dots, K_{mr}$, finally keeping the configuration yielding the minimum error on the test data set. This approach is usually referred to as *multiple-restarting*. The definitive training procedure used in this project is given in Algorithm 1.4. It takes into consideration all the observations made in this section and is based on the Levenberg-Marquardt algorithm.

[5] Recall that for perceptrons equipped with differential activation functions, two hidden layers are sufficient to approximate *any* function.

Algorithm 1.4 Complete training algorithm adopted in our numerical tests.

function $[\mathbf{w}_{opt}, E_{opt}]$ = TRAINING($(\mathcal{N}, \mathcal{V})$, P_{tr}, P_{va}, P_{te}, d, K_{ms}, ϵ, T, K_{ea})
  $E_{opt} = \infty$
  for $j = 1, \ldots, K_{ms}$ **do**
    $t = 0$, $k = 0$, $\mu = 0.01$
    randomly generate $\mathbf{w}(0)$
    $E_{tr}(\mathbf{w}(0))$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}(0)$, P_{tr}, d)
    while $t < T$ and $E_{tr}(\mathbf{w}(t)) > \epsilon$ and $k < K_{ea}$ **do**
      $[\mathbf{w}(t+1), E_{tr}(\mathbf{w}(t+1)), \mu]$ = LMITERATION($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}(t)$, P_{tr}, $E(\mathbf{w}(t))$, d, μ)
      $E_{va}(\mathbf{w}(t+1))$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}(t+1)$, P_{va}, d)
      if $E_{va}(\mathbf{w}(t+1)) > E_{va}(\mathbf{w}(t))$ **then**
        $k \leftarrow k + 1$
      else
        $k = 0$
      end if
      $t \leftarrow t + 1$
    end while
    $\mathbf{w}_{opt}^{(j)} = \mathbf{w}(t-k)$
    $E_{te}(\mathbf{w}_{opt}^{(j)})$ = OFFLINEERROR($\mathcal{N}$, $\mathcal{V}$, $\mathbf{w}_{opt}^{(j)}$, P_{te}, d)
    if $E_{te}(\mathbf{w}_{opt}^{(j)}) < E_{opt}$ **then**
      $\mathbf{w}_{opt} = \mathbf{w}_{opt}^{(j)}$
      $E_{opt} = E_{te}(\mathbf{w}_{opt}^{(j)})$
    end if
  end for
end function

Chapter 2

Reduced basis methods for nonlinear partial differential equations

In this chapter, we combine a reduced basis (RB) strategy with neural networks for the efficient resolution of parametrized nonlinear partial differential equations (PDEs) possibly defined on domains with variable shape. As illustrative yet relevant instances of nonlinear PDEs, the nonlinear Poisson equation and the stationary incompressible Navier-Stokes equations are used as expository vehicles. While the latter feature a quadratic nonlinearity laying in the convective term, we consider instances of the Poisson equation where the nonlinearity stems from a solution-dependent viscosity (or diffusion coefficient). Throughout this work, we confine the attention to one- and two-dimensional differential problems. However, the proposed RB procedure can be extended to higher dimensions, and to other classes of either linear or nonlinear PDEs as well.

Before diving into the mathematical foundation of the proposed RB method, it is worth providing an overview of the whole numerical procedure. In doing so, we wish to provide some insights on the rational behind reduced-order modeling (ROM) and to point out the motivations for resorting to neural networks, clarifying their role within the algorithm. Furthermore, we shall start setting the notation which will be used throughout the chapter.

Let $\boldsymbol{\mu}_{ph} \in \mathscr{P}_{ph} \subset \mathbb{R}^{P_{ph}}$ and $\boldsymbol{\mu}_g \in \mathscr{P}_g \subset \mathbb{R}^{P_g}$ be, respectively, the *physical* and *geometrical* parameters characterizing the differential problem at hand. The former address material properties (e.g, the viscosity in the Poisson equation), source terms and boundary conditions, while the latter define the shape of the computational domain $\widetilde{\Omega} = \widetilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 1, 2$. Furthermore, assume both $\mathscr{P}_{ph}$ and $\mathscr{P}_g$ are compact (i.e., closed and bounded) subsets of $\mathbb{R}^{P_{ph}}$ and $\mathbb{R}^{P_g}$, respectively, and denote by $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P} = \mathscr{P}_{ph} \times \mathscr{P}_g \subset \mathbb{R}^P$, $P = P_{ph} + P_g$, the overall *input vector parameter*. For a given $\boldsymbol{\mu}$ in the parameter space $\mathscr{P}$, we seek the corresponding solution $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$ of the underlying PDE in a suitable Hilbert space $\widetilde{V}(\boldsymbol{\mu}_g)$.

When dealing with domains undergoing geometrical transformations, in the context of RB methods it is convenient to introduce a parametric map

$$\boldsymbol{\Phi} : \Omega \times \mathscr{P}_g \to \widetilde{\Omega},$$

which enables the formulation and resolution of the differential problem over a fixed, parameter-independent reference domain Ω [44], such that

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

where $u(\boldsymbol{\mu})$ represents the solution over the reference domain Ω, and lies in a suitable Hilbert space V. Then, $u(\boldsymbol{\mu})$ can be regarded as a map linking the parameter domain $\mathscr{P}$ to V, i.e.,

$$u : \mathscr{P} \to V .$$

This map defines the *solution manifold* $\mathscr{M} = \{u = u(\boldsymbol{\mu}) : \boldsymbol{\mu} \in \mathscr{P}\} \subset V$, consisting of all the solutions to the parametrized PDE for any admissible input parameter [29]. In Section 2.2, we detail how to recover the formulation over the fixed domain for the differential equations in consideration. In addition, we present therein an effective way to build the volumetric parametrization $\boldsymbol{\Phi}(\boldsymbol{\mu}_g)$ given the boundary parametrization of $\widetilde{\Omega}(\boldsymbol{\mu})$ [33].

For most applications, differential equations do not admit an analytic solution in closed form. Hence, one has to resort to some numerical scheme, e.g., the finite element (FE) method, to provide a *high-fidelity* approximation $u_h(\boldsymbol{\mu})$ of $u(\boldsymbol{\mu})$. The discrete solution u_h is sought in a finite-dimensional space $V_h \subset V$ and can be identified through the associated degrees of freedom $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$, yielded by a nonlinear algebraic system of the form

$$\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^M . \tag{2.1}$$

For the FE method, presented in Section 2.4, $\mathbf{u}_h$ collects the nodal values of the corresponing solution u_h, and therefore represents the algebraic counterpart of u_h. In the following, we shall refer indiscriminately to both u_h and $\mathbf{u}_h$ as the *full-order* or *truth* solution.

Many applications entail the repeated resolution of a parametrized PDE in various settings and over different geometrical configurations, namely, for several parameter values. Notable examples include parametric sensitivity analysis, optimal control, topology optimization, and uncertainty quantification. From a geometric standpoint, one has then to span the *discrete solution manifold* $\mathscr{M}_h = \{u_h = u_h(\boldsymbol{\mu}) : \boldsymbol{\mu} \in \mathscr{P}\} \subset V_h$. At the algebraic level this would imply the resolution of a possibly large number of nonlinear systems (2.1). Clearly, as the dimension M of the discrete space V_h increases, this direct approach becomes prohibitive in the practice, both in terms of CPU time and storage, even on massive parallel workstations [52].

In this scenario, ROM (also known as *model order reduction*) replaces the computationally expensive discrete model, encoded by the large-scale system (2.1), with a reduced problem of significant smaller dimension L, with L *independent of* M. The various ROM methods proposed in the literature differ for the way the reduced system is assembled starting from the full one. In particular, *reduced basis* (RB) methods supply a *reduced space* $V_{\mathtt{rb}} \subset V_h$, approximating the discrete solution manifold $\mathscr{M}_h$ by means of L well-chosen *basis functions* $\{\psi_1, \dots, \psi_L\} \subset V_h$, namely

$$V_{\mathtt{rb}} = \operatorname{span}\{\psi_1, \dots, \psi_L\} \quad \text{and} \quad \dim V_{\mathtt{rb}} = L .$$

Given $\boldsymbol{\mu} \in \mathscr{P}$, a *reduced solution* u_L is sought in the reduced space $V_{\mathtt{rb}}$. Denoting by $\mathbb{V} = [\boldsymbol{\psi}_1 \,|\, \dots \,|\, \boldsymbol{\psi}_L] \in \mathbb{R}^{M \times L}$ the matrix collecting the nodal evaluations of the basis functions, the (algebraic) reduced solution has the form

$$\mathbf{u}_L = \mathbb{V}\, \mathbf{u}_{\mathtt{rb}} = \sum_{i=1}^{L} u_{\mathtt{rb}}^{(i)} \boldsymbol{\psi}_i ,$$

so that

$$u_L(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{i=1}^{L} u_{\mathtt{rb}}^{(i)}(\boldsymbol{\mu}) \psi_i(\boldsymbol{x}) .$$

Here, $\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^L$ embodies the coefficients for the expansion of the reduced solution in terms of the reduced basis functions and solves the *reduced* nonlinear system

$$\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L. \tag{2.2}$$

The reduced system (2.2) typically derives from the large-scale system (2.1) by replacing $\mathbf{u}_h$ with $\mathbf{u}_L$ and enforcing, for each equation, the orthogonality of the residual to the column space Col($\mathbb{V}$) of $\mathbb{V}$ [1] [44]. Therefore, the resulting RB method belongs to the class of *projection methods*.

For the sake of numerical efficiency, the whole reduced basis procedure can be decomposed into an *offline* (parameter-independent) and *online* (parameter-dependent) stage [50]. In the former, the basis functions $\{\psi_i\}_{i=1}^L$ are carefully chosen relying on a set of high-fidelity solutions - called *snapshots* - to the parameter-dependent PDE, generated through the large-scale model (2.1). In this thesis, we refer to the well-known Proper Orthogonal Decomposition (POD) [58] algorithm for the generation of the reduced basis (see Section 2.5.1). However, there exist other methods to generate such basis, as, for example, the greedy method [30]. It is worth noticing that any reduced order strategy should cooperate with the underpinning full-order model to generate reliable results [29]. This implies that the complexity of the offline phase depends on the dimension M of the original discrete model.

Once a proper reduced space has been identified, reduced solutions for new instances of the parameter input are determined online by solving system (2.2). In case of a linear PDE with an affine dependence on the parameters, one can easily decouple the online step from the full-order numerical scheme, so that the cost of any online query is uniquely related to the dimension L of the reduced space $V_{\mathtt{rb}}$ [52]. However, for differential problems with a non-affine parametric dependence, as the ones considered in this work, one needs some further machinery to guarantee a significant computational saving with respect to the underlying full-order scheme. In this respect, to recover an affine approximation of the differential operator, the empirical interpolation method (EIM) (see, e.g., [3]) and its discrete version (DEIM) (see, e.g., [10]) represent the state-of-the-art. Although their established effectiveness, these techniques unavoidably introduce a new source of inaccuracy in the model, and as a result one may need to retain a larger number of basis functions to guarantee a given degree of precision.

During the last decade, the development of a priori and a posteriori error estimators for RB methods has received an increasing attention. Indeed, such estimators, bounding the error between the reduced solution u_L and the underlying exact solution u, play an important role in the *certification* of a reduced order method, thus enabling the user to trust the output provided by the method itself [29]. Nevertheless, estimates available in the literature mainly address linear affine problems, simpler than those tackled in this work. Only recently there have been attempts to extend existing stability and convergence theories to nonlinear, non-affine problems, e.g., the Navier-Stokes equations [52], relying on advanced functional tools. In particular, the numerical stability at the reduced level may not naturally follow from the stability of the underlying high-fidelity scheme, thus

[1] We recall that the *column space* of a matrix contains all linear combinations of the columns of the matrix itself. Then:

$$\text{Col}(\mathbb{V}) = \text{span}\{\boldsymbol{\psi}_1, \dots, \boldsymbol{\psi}_L\} \subset \mathbb{R}^M.$$

In other words, Col($\mathbb{V}$) represents the algebraic counterpart of $V_{\mathtt{rb}}$.

representing a challenge for any RB technique applied to nonlinear problems [2]. For the Navier-Stokes equations, further details on this point are provided in Section 2.5.3.

The considerations drawn so far motivate the research for alternative methods to compute a reduced basis solution in the online stage, and which could provide a *fast* response without sacrifycing the *accuracy*. For this purpose, let us observe that, at the algebraic level, the vector of Col($\mathbb{V}$) closest to the high-fidelity solution $\mathbf{u}_h(\boldsymbol{\mu})$ in the Euclidean norm $\|\cdot\|_{\mathbb{R}^M}$ is given by the projection $\mathbf{u}_h^{\mathbb{V}}(\boldsymbol{\mu})$ of $\mathbf{u}_h(\boldsymbol{\mu})$ onto Col($\mathbb{V}$), namely,

$$\mathbf{u}_h^{\mathbb{V}}(\boldsymbol{\mu}) = \underset{\mathbf{w}\in \mathrm{Col}(\mathbb{V})}{\arg\min} \left\| \mathbf{w} - \mathbf{u}_h(\boldsymbol{\mu}) \right\| = \mathbb{V}\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) = \sum_{l=1}^{L} \left(\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) \right)_j \boldsymbol{\psi}_j \,.$$

In particular, we note that

$$\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu})$$

collects the coefficients for the expansion of $\mathbf{u}_h^{\mathbb{V}}$ in the reduced basis functions $\{\boldsymbol{\psi}_1, \dots, \boldsymbol{\psi}_L\}$. Hence, the key idea is to seek an approximation $\hat{\boldsymbol{\pi}}$ to the map $\boldsymbol{\pi}$, defined as

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

The approximation might be accomplished either through *interpolation* [1, 12] or *regression* [24] of a collection of target input-output pairs $\{\boldsymbol{\mu}^{(i)}, \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^{N_{tr}}$. Here, we pursue the latter approach, employing multi-layer feedforward neural networks, presented in Chapter 1. In this way, for any $\boldsymbol{\mu} \in \mathscr{P}$ the reduced-order approximation to the $\boldsymbol{\mu}$-parametrized PDE is obtained as

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

with $\hat{\boldsymbol{\pi}}^{\mathtt{NN}}$ the input-output map established by the neural network, so that $\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu}) = \hat{\boldsymbol{\pi}}^{\mathtt{NN}}(\boldsymbol{\mu})$ represents the output provided by the network when it receives $\boldsymbol{\mu}$ as input. In other terms, the online phase reduces to the (fast) evaluation of the map $\hat{\boldsymbol{\mu}}^{\mathtt{NN}}$, without involving further algebraic operations, and so leading to a *non-intrusive* ROM method. In Section 2.6, we detail how the training of the neural network can be efficiently incorporated into the RB procedure, thus leading to the POD-NN method. Benefits and disadvantages of the proposed methodology are thoroughly discussed as well.

We now proceed with a theoretical presentation of the elements mentioned so far. First, we rigorously define the functional framework for the subsequent analyses.

2.1 Parametrized nonlinear PDEs

Recalling the input vector parameter $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P} = \mathscr{P}_{ph} \times \mathscr{P}_g$, let $\widetilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 1, 2$, be a parametrized domain [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, respectively, 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

[2] A *domain* is an open and bounded set.

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

$$\widetilde{G}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) = 0 \quad \text{in } \widetilde{V}'(\boldsymbol{\mu}_g)\,, \tag{2.3}$$

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 (2.3) to be stated in a weak (or variational) form [51]. 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 (2.3) 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

$$\widetilde{g}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = 0 \quad \forall\, v \in \widetilde{V}(\boldsymbol{\mu})\,. \tag{2.4}$$

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.

Within the wide range of PDEs which suit the formulations (2.3) and (2.4), 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 Chapter 3.

2.1.1 Nonlinear Poisson equation

Despite a rather simple form, the Poisson equation has proved itself to be effective to model steady phenomena occurring in many application fields, as, e.g., electromagnetism, heat transfer, and underground flows [43]. Throughout the text, we consider the following version of the parametrized Poisson equation for a state variable $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$:

$$\begin{cases} -\widetilde{\nabla}\cdot\big(\widetilde{k}(\widetilde{\boldsymbol{x}}, \widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu})\big) = \widetilde{s}(\widetilde{\boldsymbol{x}}; \boldsymbol{\mu}_{ph}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g)\,, \quad (2.5a)\\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{g}(\widetilde{\boldsymbol{\sigma}}; \boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}_D\,, \quad (2.5b)\\ \widetilde{k}(\widetilde{\boldsymbol{\sigma}}, \widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu})\cdot\widetilde{\boldsymbol{n}} = 0 & \text{on } \widetilde{\Gamma}_N\,. \quad (2.5c) \end{cases}$$

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

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

that is, the space of squared integrable functions whose first order (distributional) derivatives are squared integrable too and which vanish on $\widetilde{\Gamma}_D$. Multiplying (2.5a) by a *test* function $v \in \widetilde{V}$, integrating over $\widetilde{\Omega}$, and exploiting integration by parts on the left-hand side, yields:

$$\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) = \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph})\, v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g), \tag{2.6}$$

where we have omitted the dependence on the space variable $\widetilde{\boldsymbol{x}}$ for ease of notation. For the integrals in Eq. (2.6) 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 $\widetilde{g}(\boldsymbol{\mu}_{ph}) \in H^{1/2}\big(\widetilde{\Gamma}_D\big)$ [3]. We assume that such a function can be constructed, e.g., by interpolation of the boundary condition. Hence, upon defining

$$\begin{aligned} \widetilde{a}(w, v; \boldsymbol{\mu}) := &\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + \widetilde{l}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} w \cdot \widetilde{\nabla} v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \\ &+ \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + \widetilde{l}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} \widetilde{l}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall\, w, v \in \widetilde{V}(\boldsymbol{\mu}_g), \end{aligned} \tag{2.7a}$$

$$\widetilde{f}(v; \boldsymbol{\mu}) := \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph})\, v \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall\, v \in \widetilde{V}(\boldsymbol{\mu}_g), \tag{2.7b}$$

the weak formulation of problem (2.5) reads: given $\boldsymbol{\mu} \in \mathscr{P}$, find $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{a}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = \widetilde{f}(v; \boldsymbol{\mu}) \quad \forall\, v \in \widetilde{V}(\boldsymbol{\mu}_g), \tag{2.8}$$

Then, the weak solution of problem (2.5) is given by $\widetilde{u}(\boldsymbol{\mu}) + \widetilde{l}(\boldsymbol{\mu})$. Note that using a lifting function makes the formulation (2.8) *symmetric*, i.e., both the solution and the test functions are picked up from the same functional space [51].

Lastly, let us remark that the weak formulation (2.8) can be cast in the form (2.4) upon setting, for any $v \in \widetilde{V}(\boldsymbol{\mu}_g)$:

$$\langle \widetilde{G}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}); v \rangle_{\widetilde{V}, \widetilde{V}'} = \widetilde{g}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = \widetilde{a}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) - \widetilde{f}(v; \boldsymbol{\mu}).$$

[3] We recall the definition of $H^{1/2}\big(\widetilde{\Gamma}_D\big)$:

$$H^{1/2}\big(\widetilde{\Gamma}_D\big) = \left\{ v \in L^2\big(\widetilde{\Gamma}_D\big) \,:\, \exists \phi \in H^1\big(\widetilde{\Omega}\big) \; s.t. \; \phi\big|_{\widetilde{\Gamma}_D} = v \right\}.$$

2.1.2 Steady 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 $\widetilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 2, 3$ [53]. 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 we consider for our numerical tests reads:

$$
\begin{cases}
\widetilde{\nabla} \cdot \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = 0 & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g), \qquad (2.9a)\\
-\nu(\boldsymbol{\mu})\, \widetilde{\Delta} \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + (\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \widetilde{\nabla}) \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \dfrac{1}{\rho(\boldsymbol{\mu})} \widetilde{\nabla} \widetilde{p}(\boldsymbol{\mu}) = \boldsymbol{0} & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g), \qquad (2.9b)\\
\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = \widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}(\boldsymbol{\mu}_g), \qquad (2.9c)\\
\widetilde{p}(\boldsymbol{\mu}) \widetilde{\boldsymbol{n}} - \nu(\boldsymbol{\mu}) \widetilde{\nabla} \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \widetilde{\boldsymbol{n}} = \boldsymbol{0} & \text{on } \widetilde{\Gamma}_N(\boldsymbol{\mu}_g). \qquad (2.9d)
\end{cases}
$$

Here, $\widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph})$ denotes the velocity field prescribed on $\widetilde{\Gamma}_D$, while 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 (2.9) 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 to preserve a dimensionless quantity of interest; we refer the reader to Section 3.3 for a practical example.
For our purpose, it is worth noticing that the nonlinearity of problem (2.9) lies in the convective term

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

which gives a *quadratic* nonlinearity. Conversely, the other terms appearing in the momentum equation (2.9b) and in Eq. (2.9a), which enforces mass conservation [52], are linear in the solution $(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{p}(\boldsymbol{\mu}))$.

Let us now introduce the velocity space $\widetilde{X}(\boldsymbol{\mu}_g) = \big[H^1_{\widetilde{\Gamma}}\big(\widetilde{\Omega}(\boldsymbol{\mu}_g)\big)\big]^d$ and the pressure space $\widetilde{Q}(\boldsymbol{\mu}_g) = L^2\big(\widetilde{\Omega}(\boldsymbol{\mu}_g)\big)$ (i.e., the space of squared integrable functions defined over $\widetilde{\Omega}(\boldsymbol{\mu}_g)$). A possible weak formulation for the differential problem (2.9) is given by: for a fixed $\boldsymbol{\mu} \in \mathscr{P}$, find $(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{p}(\boldsymbol{\mu})) \in \widetilde{X}(\boldsymbol{\mu}_g) \times \widetilde{Q}(\boldsymbol{\mu}_g)$ such that

$$
\begin{aligned}
\widetilde{a}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{c}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{d}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{b}(\widetilde{p}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) &= \widetilde{f}_1(\boldsymbol{\chi}; \boldsymbol{\mu}), \qquad (2.10a)\\
\widetilde{b}(\xi, \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}); \boldsymbol{\mu}) &= \widetilde{f}_2(\xi; \boldsymbol{\mu}), \qquad (2.10b)
\end{aligned}
$$

for any $\boldsymbol{\chi} \in \widetilde{X}(\boldsymbol{\mu}_g)$ and $\xi \in \widetilde{Q}(\boldsymbol{\mu}_g)$. Here, the trilinear form $\widetilde{c}(\cdot,\cdot,\cdot;\boldsymbol{\mu})$, the bilinear forms $\widetilde{a}(\cdot,\cdot;\boldsymbol{\mu})$, $\widetilde{b}(\cdot,\cdot;\boldsymbol{\mu})$ and $\widetilde{d}(\cdot,\cdot;\boldsymbol{\mu})$, and the functionals $\widetilde{f}_1(\cdot;\boldsymbol{\mu})$ and $\widetilde{f}_2(\cdot;\boldsymbol{\mu})$ are defined as follows[4]:

$$
\widetilde{c}(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \big(\boldsymbol{\psi} \cdot \widetilde{\nabla}\big) \boldsymbol{\chi} \cdot \boldsymbol{\eta}\, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in \big[H^1\big(\widetilde{\Omega}(\boldsymbol{\mu}_g)\big)\big]^d, \qquad (2.11a)
$$

[4]Recall that the *double dot* product $A : B$ between two square matrices $A, B \in \mathbb{R}^{n\times n}$ is defined as $A : B := \sum_{i,j=1}^n A_{ij}\, B_{ij}$.

$$\widetilde{a}(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \nu(\boldsymbol{\mu}) \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{\nabla}\boldsymbol{\psi} \,:\, \widetilde{\nabla}\boldsymbol{\chi}\, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in \widetilde{X}(\boldsymbol{\mu}_g), \tag{2.11b}$$

$$\widetilde{b}(\boldsymbol{\psi}, \xi; \boldsymbol{\mu}) = -\frac{1}{\rho(\boldsymbol{\mu})} \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \left(\widetilde{\nabla} \cdot \boldsymbol{\psi}\right) \xi\, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi} \in \widetilde{X}(\boldsymbol{\mu}_g), \forall \xi \in \widetilde{Q}(\boldsymbol{\mu}_g), \tag{2.11c}$$

$$\widetilde{d}(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \widetilde{c}(\widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{c}(\boldsymbol{\psi}, \widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in \left[H^1\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right)\right]^d, \tag{2.11d}$$

$$\widetilde{f}_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = -\widetilde{a}(\widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) - \widetilde{c}(\widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) \qquad \forall \boldsymbol{\psi} \in \widetilde{X}(\boldsymbol{\mu}_g), \tag{2.11e}$$

$$\widetilde{f}_2(\xi; \boldsymbol{\mu}) = -\widetilde{b}(\widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) \qquad \xi \in \widetilde{Q}(\boldsymbol{\mu}_g). \tag{2.11f}$$

The function $\widetilde{\boldsymbol{l}} = \widetilde{\boldsymbol{l}}(\boldsymbol{\mu}) \in \left[H^1\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right)\right]^d$ appearing inF (2.11d), (2.11e) and (2.11f) acts as a lifting function, i.e. $\widetilde{\boldsymbol{l}}|_{\widetilde{\Gamma}_D} = \widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph})$, so that the weak solution for the boundary value problem (2.9) is obtained as $\left(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \widetilde{\boldsymbol{l}}(\boldsymbol{\mu}), \widetilde{p}(\boldsymbol{\mu})\right)$. In order to derive the weak formulation (2.10), we proceed as for the Poisson equation: we first multiply (2.10a) and (2.10b) by trial functions $\boldsymbol{\chi} \in \widetilde{X}(\boldsymbol{\mu}_g)$ and $\xi \in \widetilde{Q}(\boldsymbol{\mu}_g)$, respectively, then we integrate over $\widetilde{\Omega}(\boldsymbol{\mu}_g)$ and exploit integration by parts for the term

$$\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \left[-\nu(\boldsymbol{\mu})\widetilde{\Delta}\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \frac{1}{\rho(\boldsymbol{\mu})}\widetilde{\nabla}\widetilde{p}(\boldsymbol{\mu})\right] \cdot \boldsymbol{\chi}\, d\widetilde{\Omega}(\boldsymbol{\mu}_g),$$

thus obtaining

$$\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \left[\nu(\boldsymbol{\mu})\widetilde{\nabla}\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \,:\, \widetilde{\nabla}\boldsymbol{\chi} - \widetilde{p}(\boldsymbol{\mu})\, \widetilde{\nabla}\boldsymbol{\chi}\right] d\widetilde{\Omega}(\boldsymbol{\mu}_g) + \int_{\widetilde{\Gamma}(\boldsymbol{\mu}_g)} \left[-\nu(\boldsymbol{\mu})\left(\widetilde{\boldsymbol{n}} \cdot \widetilde{\nabla}\right)\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \boldsymbol{\chi} + \widetilde{p}(\boldsymbol{\mu})\widetilde{\boldsymbol{n}} \cdot \boldsymbol{\chi}\right] d\widetilde{\Gamma}(\boldsymbol{\mu}_g).$$

Finally, (with a slight abuse of notation) we replace $\widetilde{\boldsymbol{v}}(\boldsymbol{\mu})$ with $\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \widetilde{\boldsymbol{l}}(\boldsymbol{\mu})$ and insert the homogeneous Neumann boundary conditions into the resulting integral equations.

We conclude this section showing how the variational problem (2.10) can be recast into the general form (2.4). To this end, let $\widetilde{V}(\boldsymbol{\mu}_g) = \widetilde{X}(\boldsymbol{\mu}_g) \times \widetilde{Q}(\boldsymbol{\mu}_g)$ and $\widetilde{u}(\boldsymbol{\mu}) = (\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{p}(\boldsymbol{\mu})) \in \widetilde{V}(\boldsymbol{\mu}_g)$, and then set

$$\begin{aligned}
\langle \widetilde{G}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}); v\rangle_{\widetilde{V}',\widetilde{V}} &= \widetilde{g}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) \\
&= \widetilde{a}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{c}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{d}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{b}(\widetilde{p}(\boldsymbol{\mu}), \widetilde{\nabla} \cdot \boldsymbol{\chi}; \boldsymbol{\mu}) \\
&\quad - \widetilde{f}_1(\boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{b}(\widetilde{\nabla} \cdot \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) - \widetilde{f}_2(\xi; \boldsymbol{\mu}),
\end{aligned}$$

for any $v = (\boldsymbol{\chi}, \xi) \in \widetilde{V}(\boldsymbol{\mu}_g)$.

2.2 From the original to the 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 difference, finite element, finite volume, or spectral methods). To this end, the method relies on the combination of a collection of high-fidelity approximations $\{\widetilde{u}_h(\boldsymbol{\mu}^{(1)}), \ldots, \widetilde{u}_h(\boldsymbol{\mu}^{(N)})\}$, called *snapshots*,

corresponding to the parameter values $\{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$. However, recall that 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 $1 \le j, k \le N$ and $j \ne k$, 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) $\dim\big(\mathbf{u}_h(\boldsymbol{\mu}^{(j)})\big) = \dim\big(\mathbf{u}_h(\boldsymbol{\mu}^{(k)})\big)$, 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, e.g., the finite element 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 $\boldsymbol{\Phi} : \Omega \times \mathscr{P}_g \to \widetilde{\Omega}$ such that

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

As we will prove in Sections 2.2.1 and 2.2.2, the transformation $\boldsymbol{\Phi}$ allows to restate the general problem (2.3). 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

$$G(u(\boldsymbol{\mu}); \boldsymbol{\mu}) = 0 \quad \text{in } V'\,. \tag{2.12}$$

The weak formulation of problem (2.12) reads: given $\boldsymbol{\mu} \in \mathscr{P}$, seek $u(\boldsymbol{\mu}) \in V$ such that

$$g(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = 0 \quad \forall\, v \in V\,, \tag{2.13}$$

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 $\boldsymbol{\Phi}$, thus keeping track of the original domain $\widetilde{\Omega}(\boldsymbol{\mu}_g)$ (see Section 2.2.2). Then, the solution $\widetilde{u}(\boldsymbol{\mu})$ over the original domain $\widetilde{\Omega}(\boldsymbol{\mu}_g)$ can be recovered as

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

For any $\boldsymbol{\mu} \in \mathscr{P}$ we seek a discrete solution $u_h(\boldsymbol{\mu})$ to the problem (2.12) 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 (2.12) over Ω_h is equivalent to approximating the original problem (2.3) over the mesh $\widetilde{\Omega}_h(\boldsymbol{\mu}_g)$, given by

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

Therefore, the requirements (a) and (b) are automatically fulfilled provided the map $\boldsymbol{\Phi}(\cdot;\boldsymbol{\mu}_g)$ 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 $\boldsymbol{\Phi}$ - the boundary displacement-dependent transfinite map (BDD TM) proposed in [33] and whose construction is detailed in Section 2.2.3.

In the following subsection we introduce the mathematical tools required to re-state the weak formulations (2.8) and (2.10) over the fixed domain for the problems of interest.

2.2.1 Change of variables formulae

Let Ω and $\widetilde{\Omega}$ be open bounded subsets of $\mathbb{R}^d$, $d \geq 1$, and denote respectively by $\boldsymbol{x} = [x_1, \dots, x_d]^T$ and $\widetilde{\boldsymbol{x}} = [\widetilde{x}_1, \dots, \widetilde{x}_d]^T$ a generic point belonging to each domain. Furthermore, assume $\boldsymbol{\Phi}$ is a continuous differentiable and invertible transformation linking Ω and $\widetilde{\Omega}$,

$$\begin{aligned} \boldsymbol{\Phi} \,:\, \Omega &\to \widetilde{\Omega} \\ \boldsymbol{x} &\mapsto \widetilde{\boldsymbol{x}} = \boldsymbol{\Phi}(\boldsymbol{x})\,, \end{aligned}$$

and let $\boldsymbol{\Phi}^{-1}$ be its inverse, i.e.,

$$\begin{aligned} \boldsymbol{\Phi}^{-1} \,:\, \widetilde{\Omega} &\to \Omega \\ \widetilde{\boldsymbol{x}} &\mapsto \boldsymbol{x} = \boldsymbol{\Phi}^{-1}(\widetilde{\boldsymbol{x}})\,, \end{aligned}$$

Given an arbitrary function $\psi : \Omega \to \mathbb{R}^k$, with either $k = 1$ or $k = d$, we then let $\widetilde{\psi} : \widetilde{\Omega} \to \mathbb{R}^k$ be its transposition over $\widetilde{\Omega}$ via the map $\boldsymbol{\Phi}^{-1}$, i.e.

$$\widetilde{\psi}(\widetilde{\boldsymbol{x}}) = (\psi \circ \boldsymbol{\Phi}^{-1})(\widetilde{\boldsymbol{x}}) = \psi(\boldsymbol{\Phi}^{-1}(\widetilde{\boldsymbol{x}})) = \psi(\boldsymbol{x})\,.$$

Dealing with differential operators, we first need to relate the gradients with respect to $\widetilde{\boldsymbol{x}}$ and $\boldsymbol{x}$. Let $\widetilde{\psi} : \widetilde{\Omega} \to \mathbb{R}$ be a differentiable function, then

$$\frac{\partial \widetilde{\psi}(\widetilde{\boldsymbol{x}})}{\partial \widetilde{x}_i} = \frac{\partial \psi(\boldsymbol{\Phi}^{-1}(\boldsymbol{x}))}{\partial \widetilde{x}_i} = \sum_{j=1}^{d} \frac{\partial \psi(\boldsymbol{x})}{\partial x_j} \frac{\partial x_j}{\partial \widetilde{x}_i}\,, \quad 1 \leq i \leq d\,. \tag{2.14}$$

Introducing the operators

$$\nabla = \left[\frac{\partial}{\partial x_1}, \dots, \frac{\partial}{\partial x_d}\right]^T \quad \text{and} \quad \widetilde{\nabla} = \left[\frac{\partial}{\partial \widetilde{x}_1}, \dots, \frac{\partial}{\partial \widetilde{x}_d}\right]^T,$$

and defining the Jacobian $\mathbb{J}_{\boldsymbol{\Phi}^{-1}}(\widetilde{\boldsymbol{x}}) : \widetilde{\Omega} \to \mathbb{R}^{d\times d}$ of $\boldsymbol{\Phi}^{-1}$ via

$$\left(\mathbb{J}_{\boldsymbol{\Phi}^{-1}}(\widetilde{\boldsymbol{x}})\right)_{i,j} = \frac{\partial\left(\boldsymbol{\Phi}^{-1}(\widetilde{\boldsymbol{x}})\right)_i}{\partial \widetilde{x}_j} = \frac{\partial x_i}{\partial \widetilde{x}_j}\,, \quad 1 \leq i,\, j \leq d\,,$$

the compact form of (2.14) reads:

$$\widetilde{\nabla}\widetilde{\psi}(\widetilde{\boldsymbol{x}}) = \mathbb{J}^T_{\boldsymbol{\Phi}^{-1}}(\widetilde{\boldsymbol{x}})\nabla\psi(\boldsymbol{x})\,. \tag{2.15}$$

By the invertibility of $\boldsymbol{\Phi}$, the inverse function theorem [56] ensures that

$$\mathbb{J}_{\boldsymbol{\Phi}^{-1}}(\widetilde{\boldsymbol{x}}) = \mathbb{J}^{-1}_{\boldsymbol{\Phi}}(\boldsymbol{\Phi}^{-1}(\widetilde{\boldsymbol{x}})) = \mathbb{J}^{-1}_{\boldsymbol{\Phi}}(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \widetilde{\Omega}\,.$$

with $\mathbb{J}_{\boldsymbol{\Phi}}$ the Jacobian of $\boldsymbol{\Phi}$,

$$\big(\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{x})\big)_{i,j} = \frac{\partial\big(\boldsymbol{\Phi}(\boldsymbol{x})\big)_i}{\partial x_j} = \frac{\partial \widetilde{x}_i}{\partial x_j}, \quad 1 \le i,\, j \le d\,,$$

Then, (2.15) can also be written as

$$\widetilde{\nabla}\widetilde{\psi}(\widetilde{\boldsymbol{x}}) = \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\widetilde{\boldsymbol{x}})\nabla\psi(\boldsymbol{x})\,. \tag{2.16}$$

Another major ingredient we need is the change of variables formula for integrals defined over $\widetilde{\Omega}$, as both formulations (2.8) and (2.10) of our reference problems involve integral forms. Letting $\widetilde{\psi}$ be a continuous and integrable function over $\widetilde{\Omega}$, we have [56]:

$$\int_{\widetilde{\Omega}} \widetilde{\psi}(\widetilde{\boldsymbol{x}})\, d\widetilde{\Omega} = \int_{\Omega} \psi(\boldsymbol{x}) |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{x})|\, d\Omega\,, \tag{2.17}$$

where $|\mathbb{J}_{\boldsymbol{\Phi}}|$ denotes the absolute value of the determinant of $\mathbb{J}_{\boldsymbol{\Phi}}$.

So far, we have considered sufficiently smooth and regular functions ψ and $\widetilde{\psi}$ defined either on Ω or $\widetilde{\Omega}$, repsectively. However, both formulae (2.16) and (2.17) still hold when, e.g., $\psi \in H^1(\Omega)$ and $\widetilde{\psi} \in H^1(\widetilde{\Omega})$ [5], provided that the equality in (2.16) is understood in the $L^2(\Omega)$ sense, and that the integrals in (2.17) are Lebesgue integrals. Actually, we can further relax the assumptions for the change of variables (2.17), by requiring $\widetilde{\psi} \in L^1(\widetilde{\Omega})$. Moreover, this formula can be analogously stated also for a vectorial function $\widetilde{\boldsymbol{\psi}} \in L^1(\widetilde{\Omega})$ as follows:

$$\int_{\widetilde{\Omega}} \widetilde{\boldsymbol{\psi}}\, d\widetilde{\Omega} = \int_{\Omega} \boldsymbol{\psi} |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega\,. \tag{2.18}$$

Note that we now consider functions defined in a non-classical sense, therefore we omit the dependence on the space variables as they may not be defined pointwisely.

Upon these considerations, a straightforward combination of (2.16) with (2.17) and (2.18) yields

$$\int_{\widetilde{\Omega}} \widetilde{\nabla}\widetilde{\psi}\cdot\widetilde{\nabla}\widetilde{\chi}\, d\widetilde{\Omega} = \int_{\Omega} \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\psi\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\chi\, |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\psi}, \widetilde{\chi} \in H^1(\widetilde{\Omega})\,, \tag{2.19}$$

$$\int_{\widetilde{\Omega}} \big(\widetilde{\boldsymbol{\psi}}\cdot\widetilde{\nabla}\big)\,\widetilde{\boldsymbol{\chi}}\cdot\widetilde{\boldsymbol{\eta}}\, d\widetilde{\Omega} = \int_{\Omega} \big(\boldsymbol{\psi}\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\big)\,\boldsymbol{\chi}\cdot\boldsymbol{\eta}\, |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\boldsymbol{\psi}}, \widetilde{\boldsymbol{\chi}}, \widetilde{\boldsymbol{\eta}} \in \big[H^1(\widetilde{\Omega})\big]^d\,, \tag{2.20}$$

$$\int_{\widetilde{\Omega}} \widetilde{\nabla}\widetilde{\boldsymbol{\psi}} : \widetilde{\nabla}\widetilde{\boldsymbol{\chi}}\, d\widetilde{\Omega} = \int_{\Omega} \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\boldsymbol{\psi} : \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\boldsymbol{\chi}\, |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\boldsymbol{\psi}}, \widetilde{\boldsymbol{\chi}} \in \big[H^1(\widetilde{\Omega})\big]^d\,, \tag{2.21}$$

$$\int_{\widetilde{\Omega}} \big(\widetilde{\nabla}\cdot\widetilde{\boldsymbol{\psi}}\big)\,\widetilde{\chi}\, d\widetilde{\Omega} = \int_{\Omega} \big(\mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\cdot\boldsymbol{\psi}\big)\,\chi\, |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\boldsymbol{\psi}} \in \big[H^1(\widetilde{\Omega})\big]^d, \forall \widetilde{\chi} \in H^1(\widetilde{\Omega})\,. \tag{2.22}$$

2.2.2 The problems of interest

Exploiting the results derived in the previous section, let us now cast the weak formulations (2.8) and (2.10) for the Poisson equation and the steady Navier-Stokes equations, respectively, into the general variational problem (2.13). 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

[5] Thanks to the assumed regularity of $\boldsymbol{\Phi}$, $\widetilde{\psi} \in H^1(\widetilde{\Omega})^k$, $k = 1, d$, implies $\psi \circ \boldsymbol{\Phi}^{-1} \in H^1(\Omega)$.

conditions, respectively. Setting $V = H^1_{\Gamma_D}(\Omega)$ and combining (2.7), (2.17) and (2.19), the variational formulation of the Poisson problem (2.5) over the reference domain Ω reads: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u(\boldsymbol{\mu}) \in V$ such that

$$a(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = f(v; \boldsymbol{\mu}) \quad \forall\, v \in V\,, \tag{2.23}$$

with

$$\begin{aligned} a(w, v; \boldsymbol{\mu}) = &\int_\Omega k(w + l(\boldsymbol{\mu}); \boldsymbol{\mu})\ \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla w \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla v\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \\ &+ \int_\Omega k(w + l(\boldsymbol{\mu}); \boldsymbol{\mu})\ \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla l(\boldsymbol{\mu}) \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla v\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega\,, \end{aligned} \tag{2.24a}$$

$$f(v; \boldsymbol{\mu}) = \int_\Omega s(\boldsymbol{\mu})\ v\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega\,, \tag{2.24b}$$

for any $w,\ v \in V$ and $\boldsymbol{\mu} \in \mathscr{P}$. Let us remark that, when on a parameter-independent configuration, we can avoid to distinguish between physical and geometrical parameters, since even the latter now affect the integrands in (2.24a) and (2.24b), and not the domain of integration. Moreover, we recall that $k(\boldsymbol{x}, \cdot; \boldsymbol{\mu}) = \widetilde{k}(\boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\mu}), \cdot; \boldsymbol{\mu})$ is the diffusion coefficient, $s(\boldsymbol{x}; \boldsymbol{\mu}) = \widetilde{s}(\boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\mu}); \boldsymbol{\mu})$ is the source term, and $g(\boldsymbol{x}; \boldsymbol{\mu}) = \widetilde{g}(\boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\mu}); \boldsymbol{\mu})$ represents the state field prescribed on Γ_D. Then, in (2.24) 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 (2.5) re-stated over Ω is obtained as $u(\boldsymbol{\mu}) + l(\boldsymbol{\mu})$.

Similarly, the variational formulation over Ω of the boundary value problem (2.9) for the stationary incompressible Navier-Stokes equations follows from (2.10), (2.26), (2.20), (2.21) and (2.22). Consider the velocity space $X = \left[H^1_{\Gamma_D}(\Omega)\right]^d$ and the pressure space $Q = L^2(\Omega)$, and set $V = X \times Q$. Then, given $\boldsymbol{\mu} \in \mathscr{P}$, the problem consists in finding $u(\boldsymbol{\mu}) = (\boldsymbol{v}(\boldsymbol{\mu}), p(\boldsymbol{\mu})) \in V$ so that

$$a(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + d(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + b(p(\boldsymbol{\mu}), \nabla \cdot \boldsymbol{\chi}; \boldsymbol{\mu}) = f_1(\boldsymbol{\chi}; \boldsymbol{\mu})\,, \tag{2.25a}$$

$$b(\nabla \cdot \boldsymbol{v}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) = f_2(\xi; \boldsymbol{\mu})\,, \tag{2.25b}$$

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

$$c(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = \int_\Omega \left(\boldsymbol{\psi} \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla\right) \boldsymbol{\chi} \cdot \boldsymbol{\eta}\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \qquad \forall\, \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in \left[H^1(\Omega)\right]^d, \tag{2.26a}$$

$$a(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \nu(\boldsymbol{\mu}) \int_\Omega \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla \boldsymbol{\psi}\ :\ \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla \boldsymbol{\chi}\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \qquad \forall\, \boldsymbol{\psi}, \boldsymbol{\chi} \in X\,, \tag{2.26b}$$

$$b(\boldsymbol{\psi}, \xi; \boldsymbol{\mu}) = -\frac{1}{\rho(\boldsymbol{\mu})} \int_\Omega \left(\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu}) \nabla \cdot \boldsymbol{\psi}\right)\ \xi\ |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \qquad \forall\, \boldsymbol{\psi} \in X, \forall \xi \in Q\,, \tag{2.26c}$$

$$d(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = c(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{\psi}, \boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}); \qquad \forall\, \boldsymbol{\psi}, \boldsymbol{\chi} \in X\,, \tag{2.26d}$$

$$f_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = -a(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) - c(\boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) \qquad \forall\, \boldsymbol{\psi} \in X\,, \tag{2.26e}$$

$$f_2(\xi; \boldsymbol{\mu}) = -b(\boldsymbol{l}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) \qquad \forall\, \xi \in Q\,. \tag{2.26f}$$

In the above equations, $\boldsymbol{l}(\boldsymbol{\mu}) \in \left[H^1(\Omega)\right]^d$ 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. Then, the weak solution to (2.9) defined over the fixed domain Ω is given by $\big(\boldsymbol{v}(\boldsymbol{\mu}) + \boldsymbol{l}(\boldsymbol{\mu}), p(\boldsymbol{\mu})\big)$.

2.2.3 The boundary displacement-dependent transfinite map (BDD TM)

In our numerical tests we resort to the boundary diplacement-dependent transfinite map (BDD TM) to construct the bijection $\boldsymbol{\Phi}(\boldsymbol{\mu}_g)$ from the reference domain Ω to the class of parametrized domains $\left\{\widetilde{\Omega}(\boldsymbol{\mu}_g),\, \boldsymbol{\mu}_g \in \mathscr{P}_g\right\}$. BDD TM has been proposed in [33] as a generalization of the well-known Gordon-Hall transfinite approach and it aims at overcoming the major issues associated with previously proposed extensions to the Gordon-Hall map.

Let Ω and $\widetilde{\Omega}(\boldsymbol{\mu}_g)$ be *curved polygonal* two-dimensional domains with the same number n of edges. Denote then by Γ_i, $i = 1, \dots, n$, the i-th edge of Ω, and by $\widetilde{\Gamma}_i(\boldsymbol{\mu}_g)$ the corresponding edge of $\widetilde{\Omega}(\boldsymbol{\mu}_g)$. For both the reference and the deformed domain, suppose the edges are ordered clockwise and assume they admit parametrizations of the form

$$\begin{aligned} \boldsymbol{\psi}_i \,:\, [0,1] &\to \Gamma_i \\ t &\mapsto \boldsymbol{\psi}_i(t) \end{aligned} \qquad \text{and} \qquad \begin{aligned} \widetilde{\boldsymbol{\psi}}_i \,:\, [0,1]\times\mathscr{P}_g &\to \widetilde{\Gamma}_i \\ (t,\boldsymbol{\mu}_g) &\mapsto \widetilde{\boldsymbol{\psi}}_i(t;\boldsymbol{\mu}_g)\,, \end{aligned}$$

for $i = 1, \dots, n$. Here, t denotes the arc-length, ranging from 0 to 1. In detail, letting $\boldsymbol{x}_i$ (respectively, $\widetilde{\boldsymbol{x}}_i$) be the vertex shared by Γ_{i-1} and Γ_i (resp., $\widetilde{\Gamma}_{i-1}$ and $\widetilde{\Gamma}_i$), and $\boldsymbol{x}_{i+1}$ (resp., $\widetilde{\boldsymbol{x}}_{i+1}$) be the vertex shared by Γ_i and Γ_{i+1} (resp., $\widetilde{\Gamma}_i$ and $\widetilde{\Gamma}_{i+1}$), then $t = 0$ at $\boldsymbol{x}_i$ (resp., $\widetilde{\boldsymbol{x}}_i$) and $t = 1$ at $\boldsymbol{x}_{i+1}$ (resp., $\widetilde{\boldsymbol{x}}_{i+1}$), upon defining $\boldsymbol{x}_{n+1} = \boldsymbol{x}_1$ (resp., $\widetilde{\boldsymbol{x}}_{n+1} = \widetilde{\boldsymbol{x}}_1$) and $\boldsymbol{x}_0 = \boldsymbol{x}_n$ (resp., $\widetilde{\boldsymbol{x}}_0 = \widetilde{\boldsymbol{x}}_n$).

The BDD TM induces a non-affine parametrization of the deformed domain based on the *displacement* undergone throughout the transformation by the points on the boundary $\Gamma = \partial\Omega$ of Ω. For each edge Γ_i we introduce the *displacement function* $\boldsymbol{d}_i \,:\, [0,1]\times\mathscr{P}_g \to \mathbb{R}^2$ defined as:

$$\boldsymbol{d}_i(t;\boldsymbol{\mu}_g) = \widetilde{\boldsymbol{\psi}}_i(t;\boldsymbol{\mu}_g) - \boldsymbol{\psi}_i(t)\,, \quad \forall\, t \in [0,1]\,, \forall\, \boldsymbol{\mu}_g \in \mathscr{P}_g\,.$$

For each point on Γ_i, this function gives us the relative displacement between the new and the old position of the boundary [33]. Then, the displacement of any internal point of Ω is sought as a linear combination of the boundary displacement functions $\left\{\boldsymbol{d}_i\right\}_{i=1}^n$. For this purpose, each edge Γ_i is associated with two scalar-valued functions - a *weight function* ξ_i and a *projection function* π_i. As we shall see, these functions are computed by solving just as many elliptic problems stated on Ω, therefore *independent* of the parameter $\boldsymbol{\mu}_g$. Therefore, their resolution can be naturally incorporated into the offline stage of the proposed reduced basis framework, thus ensuring numerical efficiency.

As the name suggests, for each Γ_i, $i = 1, \dots, n$, the weight function ξ_i acts as multiplier of $\boldsymbol{d}_i$ in the non-affine map, and solves the following Laplace problem:

$$\begin{cases} \Delta\xi_i = 0 & \text{in } \Omega\,, \\ \xi_i = 1 & \text{on } \Gamma_i\,, \\ \dfrac{\partial\xi_i}{\partial n} = 0 & \text{on } \Gamma_j,\ j = i-1,\, i+1\,, \\ \xi_i = 0 & \text{on } \Gamma_j,\ j \neq i-1,\, i,\, i+1\,. \end{cases} \tag{2.27}$$

Here, $\partial/\partial n$ denotes the normal derivative, and we have implicitly assumed that if $i = n$, then $\Gamma_{n+1} = \Gamma_1$, and if $i = 1$, then $\Gamma_0 = \Gamma_n$. Whereas, the projection function π_i, $i = 1, \dots, n$ somehow *projects* any internal point onto the edge Γ_i. As ξ_i, also π_i is defined by solving a

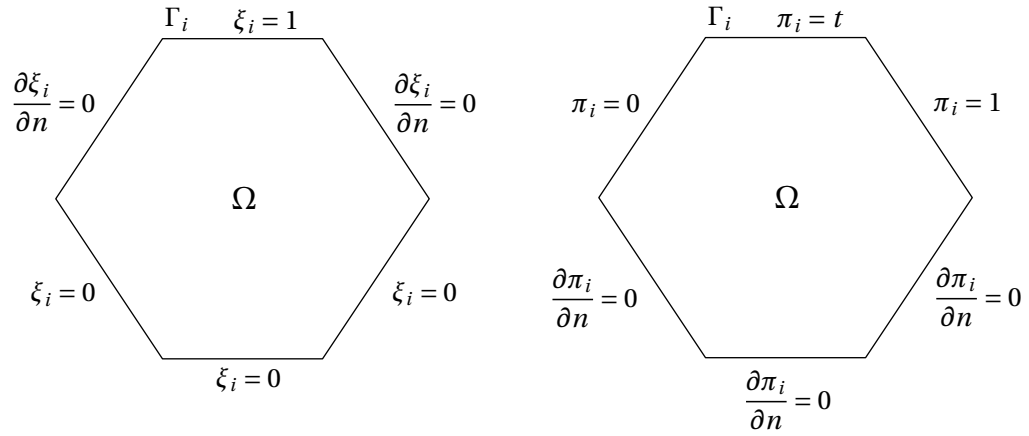


Figure 2.1. Representation of the boundary conditions for the Laplace problems (2.27) (*left*) and (2.28) (*right*) stated on a exagonal reference domain Ω.

Laplace problem over Ω, namely:

$$\begin{cases} \Delta \pi_i = 0 & \text{in } \Omega, \\ \pi_i = t & \text{on } \Gamma_i, \\ \pi_i = 0 & \text{on } \Gamma_{i-1}, \\ \pi_i = 1 & \text{on } \Gamma_{i+1}, \\ \dfrac{\partial \pi_i}{\partial n} = 0 & \text{on } \Gamma_j,\ j \neq i-1,\ i,\ i+1. \end{cases} \tag{2.28}$$

Therefore, as noted before, the computation of the functions $\{\xi_i\}_{i=1}^n$ and $\{\pi_i\}_{i=1}^n$ entails the resolution of $2n$ elliptic problems defined over the reference domain Ω; the boundary conditions for both (2.27) and (2.28) are represented in Fig. 2.1. We point out that in our numerical experiments, we let Ω be a unit square, with the bottom-left corner coinciding with the origin of the reference system, and the sides parallel to the axes. This turns out to be a convenient choice, since the Laplace problems (2.27) and (2.28) admit analytical solutions in closed form. According to the edge ordering provided in Fig. 2.2 and letting $\boldsymbol{x} = (x, y)$, we have:

$$\begin{aligned} &\xi_1 = 1 - y, \quad \xi_2 = 1 - x, \quad \xi_3 = y, \quad \xi_4 = x, \\ &\pi_1 = 1 - x, \quad \pi_2 = y, \qquad \pi_3 = x, \quad \pi_4 = 1 - y. \end{aligned} \tag{2.29}$$

Finally, given any parameter vector $\boldsymbol{\mu} \in \mathscr{P}_g$, the boundary displacement-dependent transfinite map $\boldsymbol{\Phi} : \Omega \times \mathscr{P}_g \to \widetilde{\Omega}$ is constructed as [33]:

$$\boldsymbol{\Phi}(\boldsymbol{x}; \boldsymbol{\mu}_g) = \boldsymbol{x} + \sum_{i=1}^{n} \left[\xi_i(\boldsymbol{x})\, \boldsymbol{d}_i(\pi_i(\boldsymbol{x}; \boldsymbol{\mu}_g); \boldsymbol{\mu}) - \xi_i(\boldsymbol{x})\, \xi_{i+1}(\boldsymbol{x})\, \boldsymbol{d}_i(1; \boldsymbol{\mu}_g) \right], \tag{2.30}$$

with $\xi_{n+1} = \xi_1$. Observe that, for each edge Γ_i, $i = 1, \ldots, n$, the associated displacement function is evaluated at the corresponding projection function π_i, which ranges in between 0 and 1 due to the maximum principle applied to the elliptic boundary value problem (2.28)

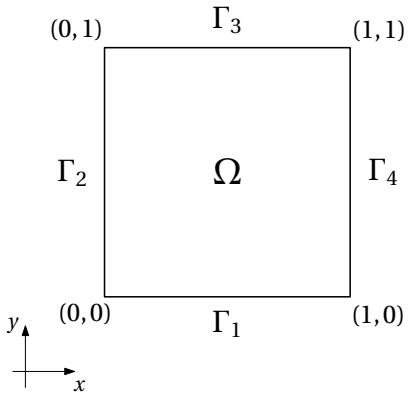


Figure 2.2. Clockwise enumeration for the edges of the reference squared domain Ω used in the numerical tests. The coordinates of the vertices are reported too.

[56]. In our case, inserting (2.29) into (2.30) yields:

$$\begin{aligned}\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu}_g) = \boldsymbol{x} &+ (1-y)\,\boldsymbol{d}_1(1-x;\boldsymbol{\mu}_g) - (1-y)\,(1-x)\,\boldsymbol{d}_1(1;\boldsymbol{\mu}_g)\\ &+ (1-x)\,\boldsymbol{d}_2(y;\boldsymbol{\mu}_g) - (1-x)\,y\,\boldsymbol{d}_2(1;\boldsymbol{\mu}_g)\\ &+ y\,\boldsymbol{d}_3(x;\boldsymbol{\mu}_g) - y\,x\,\boldsymbol{d}_3(1;\boldsymbol{\mu}_g)\\ &+ x\,\boldsymbol{d}_4(1-y;\boldsymbol{\mu}_g) - x\,(1-y)\,\boldsymbol{d}_4(1;\boldsymbol{\mu}_g)\,.\end{aligned} \tag{2.31}$$

Let us conclude this section by highlighting a couple of relevant remarks on the BDD TM (2.30), which motivate its employment within this work. In contrast to the Gordon-Hall transfinite map, the position of the reference domain does not affect the effectiveness of BDD TM [33]. Moreover, in our numerical experiments the map (2.31) yielded regular transformed grids $\widetilde{\Omega}_h(\boldsymbol{\mu}_g)$ on $\widetilde{\Omega}(\boldsymbol{\mu}_g)$. In particular, for any tested value of $\boldsymbol{\mu}_g \in \mathscr{P}$, the mesh $\widetilde{\Omega}_h(\boldsymbol{\mu}_g)$ preserved the connectivity of the reference mesh Ω_h, with no overlapping triangles. In other terms, the map (2.31) turns out to be *conformal*, so that the requirements (a) and (b) are automatically fulfilled.

2.3 Well-posedness of the test cases

Before proceeding with the discretization of the general problem (2.13) formulated over the reference domain Ω, let us briefly investigate the well-posedness for the two examples considered in this work. In the following, we simply state the main requirements on the equations and the domain which ensure the existence and uniqueness of a weak solution. For a deeper analysis and rigorous proofs, we refer the reader to the references hereunder. We should also point out that the results we provide in this section rely on the assumption of a sufficiently smooth transformation map $\boldsymbol{\Phi}(\cdot;\boldsymbol{\mu}_g)$. However, upon resorting to the boundary displacement-dependent transfinite map and a square reference domain, this condition is automatically fulfilled.

For the nonlinear Poisson problem (2.23), the solution exists and is unique provided that, for any $\boldsymbol{\mu} \in \mathscr{P}$, the viscosity $k(\cdot,\cdot;\boldsymbol{\mu}) : \Omega \times \mathbb{R} \to [0,\infty)$ is twice continuously differentiable with respect to its second argument, and, for any compact set $\Omega_c \subset \Omega$ and any bounded

interval I,

$$k(\boldsymbol{x}, r; \boldsymbol{\mu}) \geq \alpha > 0 \qquad \forall (\boldsymbol{x}, r) \in \Omega_c \times I,$$
$$\left| \frac{\partial^q k}{\partial r^q}(\boldsymbol{x}, r; \boldsymbol{\mu}) \right| < \gamma_q \qquad \text{for } q = 0, 1, 2, \ \forall (\boldsymbol{x}, r) \in \Omega_c \times I.$$

Here, α, γ_0, γ_1 and γ_2 are real positive constants. Moreover, we recall that, for the weak formulation to be well-defined, $s \in L^2(\Omega)$ and $g_D \in H^{1/2}(\Gamma_D)$ must hold as well. A complete proof of this result is offered in, e.g., [11].

Conversely, to ensure the existence of a weak solution over Ω for the boundary value problem (2.9) for the Navier-Stokes equations, we simply require the domain Ω to be Lipschitz [52]. Whereas, uniqueness is guaranteed upon a *small data* hypothesis on the boundary conditions and a possible forcing term [19].

2.4 Finite element method

In this section, we address the discretization of the parametrized problem (2.13) via the finite element (FE) method. Here, the basic ideas behind the FE strategy, an analysis of the well-posedness of the discrete problem, and a derivation of the algebraic form of the method are provided. For a comprehensive and detailed overview on finite elements, we refer the reader to, e.g., [51].

Let $V_h \subset V$ be a finite-dimensional subspace of V of dimension M. The FE approximation of the weak problem (2.12) can be cast in the form: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u_h(\boldsymbol{\mu}) \in V_h$ such that

$$g(u_h(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) = 0 \quad \forall v_h \in V_h. \tag{2.32}$$

The discretization (2.32) is known as *Galerkin approximation*, and therefore $u_h(\boldsymbol{\mu})$ is referred to as the *Galerkin solution* to the problem (2.12). To study the well-posedness of (2.32), consider the map $G : V \times \mathscr{P} \to V'$ representing our (nonlinear) differential operator, and denote by $D_u G(z, \boldsymbol{\mu}) : V \to V'$ its partial Frechét derivative at $(z, \boldsymbol{\mu}) \in V \times \mathscr{P}$, i.e., a linear bounded operator such that [56]

$$\lim_{\delta z \to 0} \frac{\left\| G(z + \delta z; \boldsymbol{\mu}) - G(z; \boldsymbol{\mu}) - D_u G(z, \boldsymbol{\mu}) \delta z \right\|_{V'}}{\|\delta z\|_V} = 0.$$

Here, $\|\cdot\|_V$ and $\|\cdot\|_{V'}$ denote suitable norms over V and V', respectively. Moreover, we denote by

$$dg[z](w, v; \boldsymbol{\mu}) = \langle D_u G(z, \boldsymbol{\mu}) w, v \rangle \quad \forall w, v \in V$$

the partial Frechét derivative of $g(\cdot, \cdot; \boldsymbol{\mu})$ with respect to its first argument and evaluated at $z \in V$. Then, the Galerkin problem (2.32) admits a unique solution $u_h(\boldsymbol{\mu}) \in V_h$ provided that the map $dg[u_h(\boldsymbol{\mu})](\cdot, \cdot; \boldsymbol{\mu})$ is *continuous*, i.e., there exists a constant $\gamma_0 < \infty$ such that

$$\gamma(\boldsymbol{\mu}) = \sup_{w_h \in V_h} \sup_{v_h \in V_h} \frac{dg[u_h(\boldsymbol{\mu})](w_h, v_h; \boldsymbol{\mu})}{\|w_h\|_V \|v_h\|_V} \leq \gamma_0, \tag{2.33}$$

and *weakly coercive* (or *inf-sup stable*), i.e., there exists a constant $\beta_0 > 0$ such that

$$\beta(\boldsymbol{\mu}) = \inf_{w_h \in V_h} \sup_{v_h \in V_h} \frac{dg[u_h(\boldsymbol{\mu})](w_h, v_h; \boldsymbol{\mu})}{\|w_h\|_V \|v_h\|_V} \geq \beta_0. \tag{2.34}$$

In the following, we generally assume that both (2.33) and (2.34) hold; yet in Section 2.4.2, we further investigate the implications of the latter requirement, known as the *inf-sup condition*, on the design of a stable finite element solver for the Navier-Stokes equations.

In order to solve the nonlinear Galerkin problem (2.32), one has to resort to some iterative method, e.g., Newton's method. Starting from an initial guess $u_h^0(\boldsymbol{\mu})$, we construct a collection of approximations $\{u_h^k(\boldsymbol{\mu})\}_{k\geq 0}$ to the Galerkin solution $u_h(\boldsymbol{\mu})$ by iteratively solving the linearized problems

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

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 $\{\phi_1, \dots, \phi_M\}$ 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.,

$$u_h(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu})\, \phi_j(\boldsymbol{x}). \tag{2.35}$$

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 (2.32) is equivalent to: given $\boldsymbol{\mu} \in \mathscr{P}$, find $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$ such that

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

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

$$\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^M, \tag{2.36}$$

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

$$\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu})\big)_i = g\left(\sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu})\, \phi_j, \phi_i; \boldsymbol{\mu}\right), \quad i = 1, \dots, M. \tag{2.37}$$

Then, for $k \geq 0$, the k-th iteration of Newton's method applied to the system (2.36) entails the resolution of the *linear* system

$$\mathbb{J}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big)\, \delta\mathbf{u}_h^k(\boldsymbol{\mu}) = -\mathbf{G}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big), \quad \delta\mathbf{u}_h^k(\boldsymbol{\mu}) \in \mathbb{R}^M. \tag{2.38}$$

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 $dg[z](\cdot,\cdot;\boldsymbol{\mu})$ (resulting from the linearity of the partial Frechét derivative $D_u G(z, \boldsymbol{\mu})$ of $G(\cdot;\boldsymbol{\mu})$), $\mathbb{J}_h(\cdot;\boldsymbol{\mu})$ is defined as

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

As mentioned above, the finite element method fits the discrete and algebraic framework presented so far, entailing a precise choice for the discrete space V_h. In this respect, consider a mesh Ω_h discretizing the domain $\Omega \subset \mathbb{R}^d$ via non-overlapping segments ($d = 1$) or triangles

($d = 2$). For each element K in the mesh, we denote by h_K its diameter and we set $h = \max_{K\in\Omega_h} h_K$, thus motivating the subscript h used so far to decorate any discrete variable. The FE spaces consist of globally continuous functions which are polynomial of degree r, $r \geq 1$, over each element, namely

$$X_h^r = \left\{ v_h \in C^0(\overline{\Omega}) \,:\, v_h\big|_K \in \mathbb{P}^r(K) \quad \forall K \in \Omega_h \right\}, \quad r = 1, 2, \dots,$$

with $\mathbb{P}^r(K)$ the space of polynomials of order r over the element K. Note that $X_h^r \subset H^1(\Omega)$ for any r, since any $v_h \in X_h^r$ is continuous and non-differentiable in a finite number of points [52]. Therefore, an approximation to the unknown field

$$u = (u_1, \dots, u_S) \in V = V_1 \times \dots \times V_S \subset \left[H^1(\Omega)\right]^S \tag{2.39}$$

satisfying the differential problem (2.13) is sought in the discrete space

$$V_h = \left(V_1 \cap X_h^{r_1}\right) \times \dots \times \left(V_S \cap X_h^{r_S}\right). \tag{2.40}$$

In other terms, we seek an approximation to the generic scalar variable u_k, $k = 1, \dots, S$, in a FE space properly modified to take the boundary conditions into account. It is worth pointing out that such an approximation can be made as accurate as desired, either decreasing h (i.e., refining the underlying mesh) or increasing the order r_k of the interpolating polynomials, possibly through adaptive strategies [51].

To define a convenient basis for X_h^r, we introduce a set of points $\{\boldsymbol{N}_i\}_{i=1}^M$, called *nodes*, which form a superset of the *vertices* of Ω_h. Upon requiring that

$$\phi_j(\boldsymbol{N}_i) = \delta_{ij} \quad \forall i, j = 1, \dots, M,$$

with δ_{ij} the Kronecker symbol, each basis function ϕ_j is characterized by a *local* support, overlapping the supports of a small number of other basis functions. Moreover

$$v_h(\boldsymbol{N}_i) = v_h^{(i)} \qquad \forall i = 1, \dots, M, \forall v_h \in X_h^r.$$

In the following, we limit ourselves to either *linear* ($r = 1$) or *quadratic* ($r = 2$) finite elements. In the former case, the nodes coincide with the vertices of Ω_h, while in the latter the vertices are augmented with the mid-points of the edges in the mesh.

Below, we give further details on the algebraic formulation of the FE discretization for the Poisson equation and the steady Navier-Stokes equations.

2.4.1 Nonlinear Poisson equation

Let $V_h = X_h^1 \cap H^1_{\Gamma_D}(\Omega)$. Recalling (2.24), the Frechét derivative $dg[z](\cdot,\cdot;\boldsymbol{\mu})$ of $g(\cdot,\cdot;\boldsymbol{\mu})$ is given by

$$\begin{aligned} dg[z](w, v; \boldsymbol{\mu}) = \int_\Omega \big[& k_r\big(z + l(\boldsymbol{\mu}); \boldsymbol{\mu}\big) \mathbb{J}_{\boldsymbol{\Phi}}^{-T} \nabla w \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T} \nabla v \; z + k\big(z + l(\boldsymbol{\mu}); \boldsymbol{\mu}\big) \mathbb{J}_{\boldsymbol{\Phi}}^{-T} \nabla z \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T} \nabla v \\ & + k_r\big(z + l(\boldsymbol{\mu}); \boldsymbol{\mu}\big) \mathbb{J}_{\boldsymbol{\Phi}}^{-T} \nabla l(\boldsymbol{\mu}) \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T} \nabla v \; w \big] \, d\Omega, \end{aligned}$$

where $k_r(\cdot,\cdot;\boldsymbol{\mu}) : \Omega \times \mathbb{R} \to \mathbb{R}$ is the partial derivative of $k(\cdot,\cdot;\boldsymbol{\mu})$ with respect to its second argument. Then, introducing the discrete lifting function $l_h(\boldsymbol{x};\boldsymbol{\mu}) \in X_h^1$, obtained via, e.g.,

Algorithm 2.1 Newton's method applied to the nonlinear system (2.36).

function $\mathbf{u}_h = \text{NEWTON}(\boldsymbol{\mu},\ \Omega_h,\ \mathbf{u}_h^0,\ \{\xi_i\}_{i=1}^n,\ \{\pi_i\}_{i=1}^n,\ \delta,\ K_{max})$
  build the transformation map $\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu})$ via Eq. (2.30)
  $k = 0$
  do
    evaluate the residual vector $\mathbf{G}_h\big(\mathbf{u}_h^k;\boldsymbol{\mu}\big)$
    form the Jacobian $\mathbb{J}_h\big(\mathbf{u}_h^k;\boldsymbol{\mu}\big)$
    solve $\mathbb{J}_h\big(\mathbf{u}_h^k;\boldsymbol{\mu}\big)\,\delta\mathbf{u}_h^k = -\mathbf{G}_h\big(\mathbf{u}_h^k;\boldsymbol{\mu}\big)$
    set $\mathbf{u}_h^{k+1} = \mathbf{u}_h^k + \delta\mathbf{u}_h^k$
    $k \leftarrow k+1$
  while $k < K_{max}$ and $\big\|\mathbf{G}_h\big(\mathbf{u}_h^k\big)\big\| > \delta$
  $\mathbf{u}_h = \mathbf{u}_h^k$
end function

interpolation of the boundary conditions over the linear finite element space X_h^1, for any $\boldsymbol{\mu} \in \mathscr{P}$ the residual vector $\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})$ reads

$$\begin{aligned}\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_i = \sum_{K\in\Omega_h}\Big\{\sum_{j=1}^n u_h^{(j)}(\boldsymbol{\mu})\int_K k\big(u_h(\boldsymbol{\mu})+l_h(\boldsymbol{\mu});\boldsymbol{\mu}\big)\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_j\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK\\ +\int_K k\big(u_h(\boldsymbol{\mu})+l_h(\boldsymbol{\mu});\boldsymbol{\mu}\big)\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla l_h(\boldsymbol{\mu})\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK\\ -\int_K s(\boldsymbol{\mu})\phi_i\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK\Big\},\end{aligned} \tag{2.41}$$

for all $i = 1,\dots,M$, with its Jacobian $\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})$ given by

$$\begin{aligned}\big(\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_{i,j} = \sum_{K\in\Omega_h}\Big\{\sum_{l=1}^M u_h^{(l)}\int_K k_r\big(u_h(\boldsymbol{\mu})+l_h(\boldsymbol{\mu});\boldsymbol{\mu}\big)\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_l\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\,\phi_j\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK\\ +\int_K k\big(u_h(\boldsymbol{\mu})+l_h(\boldsymbol{\mu});\boldsymbol{\mu}\big)\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_j\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK\\ +\int_K k_r\big(u_h(\boldsymbol{\mu})+l_h(\boldsymbol{\mu});\boldsymbol{\mu}\big)\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla l_h(\boldsymbol{\mu})\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\,\phi_j\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK\Big\},\end{aligned} \tag{2.42}$$

for any $i,\ j = 1,\dots,M$. Observe that in (2.41) and (2.42) any global integral has been split in the sum of the integrals over each element. Indeed, FE functions are continuously differentiable (actually C^∞) on each element but not globally, due to their piecewise definition. Moreover, one typically needs a quadrature rule to compute the integrals, due to the general nonlinearity of the diffusion coefficient k and the transformation map $\boldsymbol{\Phi}$. Clearly, these considerations hold also for the FE discretization of the Navier-Stokes equations.

2.4.2 Steady Navier-Stokes equations

It is well-known that for the Navier-Stokes equations a suitable choice of the FE spaces is crucial to fulfill the inf-sup stability condition (2.34) [53], which for the particular case of

the Navier-Stokes equations can be expressed as

$$\beta(\boldsymbol{\mu}) = \inf_{q_h \in Q_h} \sup_{w_h \in X_h} \frac{b_h(q_h, w_h; \boldsymbol{\mu})}{\|w_h\|_X \|v_h\|_Q} \geq \beta_0 \quad \forall \boldsymbol{\mu} \in \mathscr{P}, \tag{2.43}$$

where $X_h \subset X$ and $Q_h \subset Q$ are finite-dimensional approximations of the Hilbert spaces X and Q, equipped with the norms $\|\cdot\|_X$ and $\|\cdot\|_Q$, respectively, and $b_h(\cdot,\cdot;\boldsymbol{\mu})$ represents a suitable discretization of the bilinear form $b(\cdot,\cdot;\boldsymbol{\mu})$ (see Eq. (2.26c)). Since the differential form (2.9) is second order in the velocity and first order in the pressure, this suggests that the discretization of the velocity field should be richer than that one of the pressure distribution [51]. Therefore, a common and effective choice consists in using quadratic finite elements for the components v_x and v_y of the velocity and linear finite elements for the pressure, leading to the so called $\mathbb{P}^2 - \mathbb{P}^1$ (or Taylor-Hood) FE discretization [49]. Therefore, according to the notation introduced in (2.39) and (2.40), we have

$$u = \left(v_x, v_y, p\right) \quad \text{and} \quad V_h = \left(X_h^2 \cap H^1_{\Gamma_D}(\Omega)\right) \times \left(X_h^2 \cap H^1_{\Gamma_D}(\Omega)\right) \times \left(X_h^1 \cap L^2(\Omega)\right).$$

Let $\left\{\boldsymbol{V}_i\right\}_{i=1}^{M_v}$ be the set of vertices, associated with the linear FE basis functions $\left\{\phi_i^p\right\}_{i=1}^{M_v}$ used to discretize p, and $\left\{\boldsymbol{N}_i\right\}_{i=1}^{M_n}$ the set of nodes, associated with the quadratic FE basis functions $\left\{\phi_i^{\boldsymbol{v}}\right\}_{i=1}^{M_n}$ used to discretize $\boldsymbol{v}$. Then, setting $M = 2M_n + M_v$ and introducing the base for V_h

$$\left\{ \begin{bmatrix} \phi_1^{\boldsymbol{v}} \\ 0 \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} \phi_{M_n}^{\boldsymbol{v}} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \phi_1^{\boldsymbol{v}} \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ \phi_{M_n}^{\boldsymbol{v}} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ \phi_1^{p} \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ \phi_{M_v}^{p} \end{bmatrix} \right\},$$

for a given $\boldsymbol{\mu} \in \mathscr{P}$, the FE solution $u_h(\boldsymbol{x};\boldsymbol{\mu})$ has the form

$$u_h(\boldsymbol{x};\boldsymbol{\mu}) = \begin{bmatrix} v_{x,h}(\boldsymbol{x};\boldsymbol{\mu}) \\ v_{y,h}(\boldsymbol{x};\boldsymbol{\mu}) \\ p_h(\boldsymbol{x};\boldsymbol{\mu}) \end{bmatrix} = \sum_{j=1}^{M_n} v_{x,h}^{(j)}(\boldsymbol{\mu}) \begin{bmatrix} \phi_j^{\boldsymbol{v}}(\boldsymbol{x}) \\ 0 \\ 0 \end{bmatrix} + \sum_{j=1}^{M_n} v_{y,h}^{(j)}(\boldsymbol{\mu}) \begin{bmatrix} 0 \\ \phi_j^{\boldsymbol{v}}(\boldsymbol{x}) \\ 0 \end{bmatrix} + \sum_{j=1}^{M_v} p_h^{(j)}(\boldsymbol{\mu}) \begin{bmatrix} 0 \\ 0 \\ \phi_j^{p}(\boldsymbol{x}) \end{bmatrix}.$$

As usual, we denote by $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$ the vector collecting the degrees of freedom of $u_h(\boldsymbol{\mu})$, ordered as follows:

$$\mathbf{u}_h(\boldsymbol{\mu}) = \big[\underbrace{v_{x,h}^{(1)}(\boldsymbol{\mu}), \dots, v_{x,h}^{(M_n)}(\boldsymbol{\mu}), v_{y,h}^{(1)}(\boldsymbol{\mu}), \dots, v_{y,h}^{(M_n)}(\boldsymbol{\mu})}_{\mathbf{v}_h(\boldsymbol{\mu})^T \in \mathbb{R}^{2M_n}}, \underbrace{p_h^{(1)}(\boldsymbol{\mu}), \dots, p_h^{(M_v)}(\boldsymbol{\mu})}_{\mathbf{p}_h(\boldsymbol{\mu})^T \in \mathbb{R}^{M_v}} \big]^T.$$

Moreover, in view of the following computations, let us also introduce a basis $\left\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{2M_n}\right\}$ for the velocity trial and test space, with, for $j = 1, \dots, M_n$,

$$\boldsymbol{\phi}_j = \begin{bmatrix} \phi_j^{\boldsymbol{v}} \\ 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{\phi}_{M_n+j} = \begin{bmatrix} 0 \\ \phi_j^{\boldsymbol{v}} \end{bmatrix},$$

so that

$$\begin{aligned} \boldsymbol{v}_h(\boldsymbol{x};\boldsymbol{\mu}) &= \sum_{j=1}^{M_n} \left(v_{x,h}^{(j)}(\boldsymbol{\mu})\, \boldsymbol{\phi}_j(\boldsymbol{x}) + v_{y,h}^{(j)}(\boldsymbol{\mu})\, \boldsymbol{\phi}_{M_n+j}(\boldsymbol{x}) \right) \\ &= \sum_{j=1}^{M_n} \left(u_h^{(j)}(\boldsymbol{\mu})\, \boldsymbol{\phi}_j(\boldsymbol{x}) + u_h^{(M_n+j)}(\boldsymbol{\mu})\, \boldsymbol{\phi}_{M_n+j}(\boldsymbol{x}) \right). \end{aligned}$$

Given (2.25), (2.26) and (2.37), the residual vector for the Navier-Stokes equations is defined as (we omit the dependence on $\boldsymbol{\mu}$ to ease the notation):

$$\begin{aligned}\big(\mathbf{G}_h(\mathbf{u}_h;\boldsymbol{\mu})\big)_i &= \sum_{j=1}^{2M_n} v_h^{(j)}\left[a_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu})+d_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu})\right]+\sum_{j=1}^{2M_n}\sum_{k=1}^{2M_n} v_h^{(j)}v_h^{(k)}c_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_k,\boldsymbol{\phi}_i;\boldsymbol{\mu})\\ &\quad+\sum_{q=1}^{M_v} p_h^{(q)}b_h(\phi_q^p,\boldsymbol{\phi}_i;\boldsymbol{\mu})-f_{1h}(\boldsymbol{\phi}_i;\boldsymbol{\mu}) \qquad \text{for } i=1,\dots 2M_n,\\ \big(\mathbf{G}_h(\mathbf{u}_h;\boldsymbol{\mu})\big)_{2M_n+q} &= \sum_{j=1}^{2M_n} v_h^{(j)}b_h(\phi_q^p,\boldsymbol{\phi}_j,;\boldsymbol{\mu})-f_{2h}(\phi_q^p;\boldsymbol{\mu}) \qquad \text{for } q=1,\dots M_v,\end{aligned}$$

with

$$\begin{aligned}a_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu}) &= \sum_{K\in\Omega_h} a_K(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu}), & d_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu}) &= \sum_{K\in\Omega_h} d_K(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu}),\\ c_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_k,\boldsymbol{\phi}_i;\boldsymbol{\mu}) &= \sum_{K\in\Omega_h} c_K(\boldsymbol{\phi}_j,\boldsymbol{\phi}_k,\boldsymbol{\phi}_i;\boldsymbol{\mu}), & b_h(\phi_q^p,\boldsymbol{\phi}_i;\boldsymbol{\mu}) &= \sum_{K\in\Omega_h} b_K(\phi_q^p,\boldsymbol{\phi}_i;\boldsymbol{\mu}),\\ f_{1h}(\boldsymbol{\phi}_i;\boldsymbol{\mu}) &= \sum_{K\in\Omega_h} f_{1K}(\boldsymbol{\phi}_i;\boldsymbol{\mu}), & f_{2h}(\phi_q^p;\boldsymbol{\mu}) &= \sum_{K\in\Omega_h} f_{2K}(\phi_q^p;\boldsymbol{\mu}),\end{aligned}$$

where $a_K(\cdot,\cdot;\boldsymbol{\mu})$, $d_K(\cdot,\cdot;\boldsymbol{\mu})$, $c_K(\cdot,\cdot,\cdot;\boldsymbol{\mu})$, $b_K(\cdot,\cdot;\boldsymbol{\mu})$, $f_{1K}(\cdot,\cdot;\boldsymbol{\mu})$ and $f_{2K}(\cdot,\cdot;\boldsymbol{\mu})$ represent the restrictions of $a(\cdot,\cdot;\boldsymbol{\mu})$, $d(\cdot,\cdot;\boldsymbol{\mu})$, $c(\cdot,\cdot,\cdot;\boldsymbol{\mu})$, $b(\cdot,\cdot;\boldsymbol{\mu})$, $f_1(\cdot,\cdot;\boldsymbol{\mu})$ and $f_2(\cdot,\cdot;\boldsymbol{\mu})$, respectively, over the element $K\in\Omega_h$. Moreover, we recall that the definitions of the bilinear form $d(\cdot,\cdot;\boldsymbol{\mu})$ and the linear forms $f_1(\cdot;\boldsymbol{\mu})$ and $f_2(\cdot;\boldsymbol{\mu})$ rely on the introduction of a lifting vector $\boldsymbol{l}=\boldsymbol{l}(\boldsymbol{\mu})\in\left[H^1(\Omega)\right]^2$, which has then to be interpolated over the finite element space $\left[X_h^2\right]^2$, yielding the discrete lifting vector $\boldsymbol{l}_h=\boldsymbol{l}_h\in\left[X_h^2\right]^2$.
Upon introducing the matrices $\mathbb{A}(\boldsymbol{\mu})\in\mathbb{R}^{2M_n\times 2M_n}$, $\mathbb{C}(\mathbf{v}_h(\boldsymbol{\mu});\boldsymbol{\mu})\in\mathbb{R}^{2M_n\times 2M_n}$ and $\mathbb{B}(\boldsymbol{\mu})\in\mathbb{R}^{M_v\times 2M_n}$, and the vectors $\mathbf{f}_1(\boldsymbol{\mu})\in\mathbb{R}^{2M_n}$ and $\mathbf{f}_2(\boldsymbol{\mu})\in\mathbb{R}^{M_v}$, defined as

$$\begin{aligned}\big(\mathbb{A}(\boldsymbol{\mu})\big)_{i,j} &= a_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu})+d_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_i;\boldsymbol{\mu}) && i,j=1,\dots,2M_n,\\ \big(\mathbb{C}(\mathbf{v}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_{i,j} &= \sum_{k=1}^{2M_n} v_h^{(k)}c_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_k,\boldsymbol{\phi}_i;\boldsymbol{\mu}) && i,j=1,\dots,2M_n,\\ \big(\mathbb{B}(\boldsymbol{\mu})\big)_{q,j} &= b_h(\phi_q^p,\nabla\cdot\boldsymbol{\phi}_j;\boldsymbol{\mu}) && q=1,\dots,M_v,\ j=1,\dots,2M_n,\\ \big(\mathbf{f}_1(\boldsymbol{\mu})\big)_i &= f_{1h}(\boldsymbol{\phi}_i;\boldsymbol{\mu}) && i=1,\dots,2M_n,\\ \big(\mathbf{f}_2(\boldsymbol{\mu})\big)_q &= f_{2h}(\phi_p^q;\boldsymbol{\mu}) && q=1,\dots,M_v,\end{aligned}$$

the nonlinear system obtained by the FE method can be written in compact form as follows:

$$\begin{bmatrix}\mathbb{A}(\boldsymbol{\mu})+\mathbb{C}(\mathbf{v}_h(\boldsymbol{\mu});\boldsymbol{\mu}) & \mathbb{B}(\boldsymbol{\mu})^T\\ \mathbb{B}(\boldsymbol{\mu}) & 0\end{bmatrix}\begin{bmatrix}\mathbf{v}_h(\boldsymbol{\mu})\\ \mathbf{p}_h(\boldsymbol{\mu})\end{bmatrix}=\begin{bmatrix}\mathbf{f}_1(\boldsymbol{\mu})\\ \mathbf{f}_2(\boldsymbol{\mu})\end{bmatrix}. \tag{2.44}$$

Hence, the Jacobian is given by:

$$\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})=\begin{bmatrix}\mathbb{A}(\boldsymbol{\mu})+\left(\bar{\mathbb{C}}(\boldsymbol{\mu})+\bar{\mathbb{C}}(\boldsymbol{\mu})^T\right)\mathbf{v}_h(\boldsymbol{\mu}) & \mathbb{B}^T(\boldsymbol{\mu})\\ \mathbb{B}(\boldsymbol{\mu}) & 0\end{bmatrix},$$

where $\bar{\mathbb{C}}(\boldsymbol{\mu})\in\mathbb{R}^{2M_n\times 2M_n}$ is defined as

$$\big(\bar{\mathbb{C}}(\boldsymbol{\mu})\big)_{i,j}=\sum_{k=1}^{2M_n} c_h(\boldsymbol{\phi}_j,\boldsymbol{\phi}_k,\boldsymbol{\phi}_i;\boldsymbol{\mu}) \quad \text{for } i,j=1,\dots,2M_n.$$

2.5 POD-Galerkin RB method

As detailed in the previous section, the finite element discretization of the $\boldsymbol{\mu}$-dependent nonlinear differential problem (2.13), combined with Newton's method, entails the assembly and resolution of (possibly) many linear systems of the form (2.38), whose dimension is directly related to (i) the size of the underlying grid and (ii) the order of the polynomial 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, even resorting to high-performance parallel workstations, this approach is hardly affordable in *many-query* and *real-time* contexts, where one is interested in a fast and reliable prediction of an *output of interest*, i.e, a functional of the field variable $u(\boldsymbol{\mu})$, for many instances of $\boldsymbol{\mu} \in \mathscr{P}$ [16]. 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 [29, 52].

In this work we focus on reduced basis (RB) methods for the approximation of the variational problem (2.13). To this end, let us recall the definition of the solution manifold $\mathscr{M}$,

$$\mathscr{M} = \{u(\boldsymbol{\mu}) \,:\, \boldsymbol{\mu} \in \mathscr{P}\} \subset V\,,$$

and its discrete counterpart $\mathscr{M}_h$,

$$\mathscr{M}_h = \{u_h(\boldsymbol{\mu}) \,:\, \boldsymbol{\mu} \in \mathscr{P}\} \subset V_h\,.$$

For any $\boldsymbol{\mu} \in \mathscr{P}$, we assume that the FE solution $u_h(\boldsymbol{\mu})$ can be lead as close as desired (in the V-norm) to the corresponding continuous solution $u_h(\boldsymbol{\mu})$ (either by refining the computational mesh or by increasing the order of the FE space), so that $\mathscr{M}_h$ provides a good approximation of $\mathscr{M}$. Hence, in the following we refer to $u_h(\boldsymbol{\mu})$ as the *truth* solution.

Reduced basis methods seek an approximated solution to the problem (2.13) as a linear combination of parameter-independent functions $\{\psi_1, \dots, \psi_L\} \subset V_h$, called *reduced basis functions*, built from a collection of high-fidelity solutions $\{u_h(\boldsymbol{\mu}^{(1)}), \dots, u_h(\boldsymbol{\mu}^{(N)})\} \subset \mathscr{M}_h$, called *snapshots*, where the discrete and finite set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathscr{P}$ may consist of either a uniform lattice or randomly generated points over the parameter domain $\mathscr{P}$ [29]. 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 [58], 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.,

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

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

$$u_L(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{l=1}^{L} u_{\mathtt{rb}}^{(l)}(\boldsymbol{\mu})\, \psi_l(\boldsymbol{x}) \in V_{\mathtt{rb}}\,, \tag{2.45}$$

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. Before further

proceeding with the derivation of the method, let us gain some insights into the rational behind the reduced basis approach. As one can foresee, RB methods (potentially) enable a relevant reduction in the computational effort when the dimension of the associated test and trial space $V_{\mathtt{rb}}$ is significantly smaller than the dimension of the original finite element space V_h; in other terms, $L << M$ must hold. However, this assumes that the solution manifold $\mathscr{M} \subset V$ (or, equivalently, $\mathscr{M}_h \subset V_h$, for what previously said) is actually of low-dimension, and can then be accurately approximated by a subspace of reduced dimension L [29]. To further investigate this necessary hypothesis, it is worth introducing the notion of Kolmogorov L-width. The definition is [40]:

Definition 2.1. *Let X be a linear space equipped with the norm $\|\cdot\|_X$, A be a subset of X and X_L be a generic L-dimensional subspace of X. The deviation of A from X_L is defined as*

$$E(A; X_L) = \sup_{x \in A} \inf_{y \in X_L} \left\| x - y \right\|_X .$$

Then, the Kolmogorov L-width *of A in X is given by*

$$\begin{aligned} d_L(A, X) &= \inf \left\{ E(A; X_L) \ : \ X_L \text{ is an } L\text{-dimensional subspace of } X \right\} \\ &= \inf_{X_L} \sup_{x \in A} \inf_{y \in X_L} \left\| x - y \right\|_X . \end{aligned} \tag{2.46}$$

Therefore, $d_L(A, X)$ measures the extent to which the subset A of the vector space X can be well-approximated by an L-dimensional subspace [40]. Indeed, there exist many situations in which the Kolmogorov L-width shows a graceful behaviour with L, e.g., an exponential decay. In our case, $X = V$ and $A = \mathscr{M}$, and we can refer to regularity of the solutions $u(\boldsymbol{\mu})$ with respect to the parameter $\boldsymbol{\mu}$, or even to analyticity in the parameter dependence [7].

Nevertheless, we still have to ensure that the RB space $V_{\mathtt{rb}}$, i.e., the chosen X_L in (2.46), attains the infimum $d_L(\mathscr{M}, V)$, or at least a value close to the infimum. At this regard, let us consider the following bound for the error committed when approximating the continuous solution $u(\boldsymbol{\mu})$ with $u_L(\boldsymbol{\mu})$:

$$\left\| u(\boldsymbol{\mu}) - u_L(\boldsymbol{\mu}) \right\|_V \leq \left\| u(\boldsymbol{\mu}) - u_h(\boldsymbol{\mu}) \right\|_V + \left\| u_h(\boldsymbol{\mu}) - u_L(\boldsymbol{\mu}) \right\|_V \quad \forall \boldsymbol{\mu} \in \mathscr{P} .$$

The first term on the right-hand side measures the discrepancy between the continuous solution and its high-fidelity approximation. For what said above, this error can be lower to any desired level of accuracy. Therefore, the reliability of the solution provided by any reduced basis technique relies on a sound control of the second term of the right-hand side $\left\| u_h(\boldsymbol{\mu}) - u_L(\boldsymbol{\mu}) \right\|_V$, i.e., the error between the truth and the reduced solution. The last decade has whitnessed the development of different *a priori* and *a posteriori* estimates for such error (see, e.g., [7, 29, 40]), thus *certifying* the RB procedure, that is, enabling the user to trust the output of the RB method. However, as already mentioned, the range of application of these estimates is usually limited to linear problems with an affine dependence on the parameters. Although recent and relevant improvements have been achieved also for the Navier-Stokes equations (see, e.g., [16, 52]), they rely on non-trivial results from functional analysis and involve rather long calculations. Hence, it is not the intent of this project to further investigate and employ these estimates. Yet, in our numerical simulations we shall *empirically* study the effectiveness of the proposed POD-Galerkin method by evaluating the error $\left\| u_h(\boldsymbol{\mu}) - u_L(\boldsymbol{\mu}) \right\|_V$ on a finite and discrete *test* dataset $\Xi_{te} \in \mathscr{P}$. We refer the reader to Section 2.6 for a deeper discussion.

Let us now resume the derivation of the POD-Galerkin RB method. To unearth $u_L(\boldsymbol{\mu})$, whose general form is given in (2.45), we proceed to project the variational problem (2.13) 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

$$g(u_L(\boldsymbol{\mu}), v_L; \boldsymbol{\mu}) = 0 \quad \forall\, v_L \in V_{\mathtt{rb}}\,. \tag{2.47}$$

Then, Newton's method applied to (2.47) 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

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

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 finite element functions. Hence, we denote by $\boldsymbol{\psi}_l \in \mathbb{R}^M$ the vector collecting the nodal values of ψ_l, for $l = 1, \dots, L$, and introduce the matrix $\mathbb{V} = \big[\boldsymbol{\psi}_1, \big| \dots \big|, \boldsymbol{\psi}_L\big] \in \mathbb{R}^{M\times L}$. 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.,

$$\mathbf{v}_L = \mathbb{V}\, \mathbf{v}_{\mathtt{rb}}\,. \tag{2.48}$$

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 (2.47), for any $\boldsymbol{\mu} \in \mathscr{P}$ we get the set of equations

$$g(u_L(\boldsymbol{\mu}), \psi_l; \boldsymbol{\mu}) = 0 \quad 1 \leq l \leq L. \tag{2.49}$$

Inserting into (2.49) 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:

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

where the last equality follows from the definition (2.37) of the residual vector $\mathbf{G}_h(\cdot;\boldsymbol{\mu})$ and the notation introduced in (2.48). Then, the algebraic formulation of the reduced basis problem (2.47) can be written in compact form as:

$$\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbb{V}\, \mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L\,. \tag{2.50}$$

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

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{w};\boldsymbol{\mu}) = \mathbb{V}^T \mathbb{J}_h(\mathbb{V}\,\mathbf{w};\boldsymbol{\mu})\,\mathbb{V} \in \mathbb{R}^{L\times L} \quad \forall \mathbf{w} \in \mathbb{R}^L,\ \forall \boldsymbol{\mu} \in \mathscr{P}. \tag{2.51}$$

Hence, starting from an initial guess $\mathbf{u}^0_{\mathtt{rb}} \in \mathbb{R}^L$, each iteration k, $k \geq 0$, of Newton's method applied to the reduced nonlinear system (2.50) entails the resolution of the linear system

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{u}^k_{\mathtt{rb}}(\boldsymbol{\mu});\boldsymbol{\mu})\ \delta\mathbf{u}^k_{\mathtt{rb}}(\boldsymbol{\mu}) = -\mathbf{G}_{\mathtt{rb}}(\mathbf{u}^k_{\mathtt{rb}}(\boldsymbol{\mu});\boldsymbol{\mu}), \tag{2.52}$$

with $\mathbf{u}^{k+1}_{\mathtt{rb}}(\boldsymbol{\mu}) = \mathbf{u}^k_{\mathtt{rb}}(\boldsymbol{\mu}) + \delta\mathbf{u}^k_{\mathtt{rb}}(\boldsymbol{\mu})$. 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.

In the upcoming subsection, we detail the construction of a reduced basis via the POD method, highlighting its optimality properties and potential disadvantages.

2.5.1 Proper Orthogonal Decomposition

In a general sense, *Proper Orthogonal Decomposition* (POD) is a powerful method of data analysis aimed at reducing the cardinality of a given high-dimensional dataset (or system). First, an orthonormal basis for the original data space is generated, consisting of basis vectors called *modes* or *principal components*. Ideally, the first modes embody much of the *energy* of the system, and so they express the *essential information* of data [52]. Therefore, a meaningful low-dimensional representation of data is obtained by truncating the orthonormal basis to retain only a few POD modes, then projecting the system onto the truncated basis [58]. This approach perfectly fits our needs, as we shall see hereunder. However, it is clear that the interest in the POD method extends far beyond the field of reduced-order modeling techniques, finding a fertile ground in, e.g., random variables, image processing, and data compression [27].

Consider a collection of N snapshots (i.e., high-fidelity solutions to the problem (2.13)) $\{u_h(\boldsymbol{\mu}^{(1)}), \dots, u_h(\boldsymbol{\mu}^{(N)})\} \subset \mathscr{M}_h$ corresponding to the finite and discrete parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathscr{P}$, and let $\mathscr{M}_{\Xi_N}$ be the subspace spanned by the snapshots, i.e.,

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

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 finite element space). Then, we aim at finding a parameter-independent *reduced basis* for $\mathscr{M}_{\Xi_N}$, i.e., a collection of FE functions $\{\psi_1, \dots, \psi_L\} \subset \mathscr{M}_{\Xi_N}$, with $L \ll M$, N, and L *independent* of both M and N, so that the associated linear space

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

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

Algorithm 2.2 Newton's method applied to the reduced nonlinear system (2.50).

function $\mathbf{u}_{\mathtt{rb}} = \text{NEWTONRB}(\boldsymbol{\mu},\ \Omega_h,\ \mathbb{V},\ \mathbf{u}^0_{\mathtt{rb}},\ \{\xi_i\}_{i=1}^n,\ \{\pi_i\}_{i=1}^n,\ \delta,\ K_{max})$
  build the transformation map $\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu})$ via Eq. (2.30)
  $k = 0$
  do
    set $\mathbf{u}_L^k = \mathbb{V}\mathbf{u}_{\mathtt{rb}}^k$
    evaluate the reduced residual vector $\mathbf{G}_{\mathtt{rb}}\big(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu}\big) = \mathbb{V}^T\mathbf{G}_h\big(\mathbf{u}_L^k;\boldsymbol{\mu}\big)$
    form the Jacobian $\mathbb{J}_{\mathtt{rb}}\big(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu}\big) = \mathbb{V}^T\mathbb{J}_h\big(\mathbf{u}_L^k;\boldsymbol{\mu}\big)\mathbb{V}$
    solve $\mathbb{J}_{\mathtt{rb}}\big(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu}\big)\,\delta\mathbf{u}_{\mathtt{rb}}^k = -\mathbf{G}_{\mathtt{rb}}\big(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu}\big)$
    set $\mathbf{u}_{\mathtt{rb}}^{k+1} = \mathbf{u}_{\mathtt{rb}}^k + \delta\mathbf{u}_{\mathtt{rb}}^k$
    $k \leftarrow k+1$
  while $k < K_{max}$ and $\big\|\mathbf{G}_{\mathtt{rb}}\big(\mathbf{u}_{\mathtt{rb}}^k\big)\big\| > \delta$
  $\mathbf{u}_{\mathtt{rb}} = \mathbf{u}_{\mathtt{rb}}^k$
end function

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

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

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

$$\mathbb{S} = \mathbb{W}\begin{bmatrix}\mathbb{D} & 0\\ 0 & 0\end{bmatrix}\mathbb{Z}^T = \mathbb{W}\,\Sigma\,\mathbb{Z}^T, \tag{2.53}$$

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 $\mathbf{w}_m \in \mathbb{R}^M$, $m = 1,\dots,M$, of $\mathbb{W}$ are called *left singular vectors* of $\mathbb{S}$, and the columns $\mathbf{z}_n \in \mathbb{R}^N$, $n = 1,\dots,N$, of $\mathbb{Z}$ are called *right singular vectors* of $\mathbb{S}$, and they are related by the following relations:

$$\mathbb{S}\mathbb{S}^T\mathbf{w}_m = \begin{cases}\sigma_m^2\,\mathbf{w}_m & \text{for } 1 \le m \le R,\\ \mathbf{0} & \text{for } R+1 \le m \le M,\end{cases} \tag{2.54}$$

$$\mathbb{S}^T\mathbb{S}\,\mathbf{z}_n = \begin{cases}\sigma_n^2\,\mathbf{z}_n & \text{for } 1 \le n \le R,\\ \mathbf{0} & \text{for } R+1 \le n \le N,\end{cases} \tag{2.55}$$

$$\mathbb{S}\,\mathbf{z}_i = \sigma_i\,\mathbf{w}_i \qquad \text{for } 1 \le i \le R, \tag{2.56}$$

$$\mathbb{S}^T\mathbf{w}_i = \sigma_i\,\mathbf{z}_i \qquad \text{for } 1 \le i \le R. \tag{2.57}$$

In particular, (2.54) and (2.55) state that the first R columns of $\mathbb{W}$ and $\mathbb{Z}$ are eigenvectors of $\mathbb{S}\mathbb{S}^T$ and $\mathbb{S}^T\mathbb{S}$, respectively, with eigenvalues $\lambda_i = \sigma_i^2$, $i = 1,\dots,R$, while the remaining columns (if any, i.e., if $R < M$ or $R < N$, respectively) belongs to the kernel of $\mathbb{S}\mathbb{S}^T$ and $\mathbb{S}^T\mathbb{S}$, respectively. Due to the sparsity pattern of Σ in (2.53), 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.,

$$\mathbb{W}_R = \left[\mathbf{w}_1 \,\middle|\, \dots \,\middle|\, \mathbf{w}_R\right] \quad \text{and} \quad \mathbb{Z}_R = \left[\mathbf{z}_1 \,\middle|\, \dots \,\middle|\, \mathbf{z}_R\right]. \tag{2.58}$$

Letting $\mathbb{B}_R = \mathbb{D}\,\mathbb{Z}_R^T \in \mathbb{R}^{R\times N}$ and exploiting the orthonormality of the columns of $\mathbb{W}_R$, the generic column $\mathbf{s}_n = \mathbf{u}_h(\boldsymbol{\mu}^{(n)})$ of $\mathbb{S}$, $n = 1, \dots N$, can be expressed as [58]:

$$\begin{aligned}\mathbf{s}_n &= \sum_{r=1}^{R} \left(\mathbb{B}_R\right)_{r,n}\mathbf{w}_r = \sum_{r=1}^{R} \left(\mathbb{D}\,\mathbb{Z}_R^T\right)_{r,n}\mathbf{w}_r = \sum_{r=1}^{R} \Big(\underbrace{\mathbb{W}_R^T \overbrace{\mathbb{W}_R}^{\mathbb{S}}}_{\mathbb{I}_R\in\mathbb{R}^{R\times R}}\,\mathbb{D}\,\mathbb{Z}_R^T\Big)_{r,n}\mathbf{w}_j \\ &= \sum_{r=1}^{R} \left(\left(\mathbb{W}_R\right)^T\mathbb{S}\right)_{r,n}\mathbf{w}_r = \sum_{r=1}^{R} \left(\mathbf{w}_r^T\mathbf{s}_n\right)\mathbf{w}_r = \sum_{r=1}^{R} (\mathbf{s}_n, \mathbf{w}_r)_{\mathbb{R}^M}\mathbf{w}_r\,,\end{aligned}$$

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 $\mathbb{S}$. Moreover, from the above calculations follow that $\left(\mathbb{B}_R\right)_{r,n} = (\mathbf{s}_n, \mathbf{w}_r)_{\mathbb{R}^M}$ for $n = 1, \dots, N$ and $r = 1, \dots, R$.

Assume 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

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

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

$$\varepsilon(\widetilde{\mathbf{w}}_1, \dots, \widetilde{\mathbf{w}}_L) = \sum_{n=1}^{N} \left\| \mathbf{s}_n - \sum_{l=1}^{L} \left(\mathbf{s}_n, \widetilde{\mathbf{w}}_l\right)_{\mathbb{R}^M}\widetilde{\mathbf{w}}_l \right\|_{\mathbb{R}^M}^2. \tag{2.59}$$

The following theorem states that the basis $\{\mathbf{w}_1, \dots, \mathbf{w}_L\}$ consisting of the first L left singular values of $\mathbb{S}$ minimizes (2.59) among all the orthonormal bases of $\mathbb{R}^L$.

Theorem 2.5.1 (Schmidt-Eckart-Young)**.** *Consider a rectangular matrix $\mathbb{S} = \left[\mathbf{s}_1 \,\middle|\, \dots \,|\, \mathbf{s}_N\right] \in \mathbb{R}^{M\times N}$ with rank $R \le \min\{M, N\}$, and let $\mathbb{S} = \mathbb{W}\,\Sigma\,\mathbb{Z}^T$ be the singular value decomposition (SVD) of $\mathbb{S}$, with $\mathbb{W} = \left[\mathbf{w}_1 \,\middle|\, \dots \,|\, \mathbf{w}_M\right] \in \mathbb{R}^{M\times M}$ and $\mathbb{Z} = \left[\mathbf{z}_1 \,\middle|\, \dots \,|\, \mathbf{z}_N\right] \in \mathbb{R}^{N\times N}$ orthogonal matrices, and $\Sigma \in \mathbb{R}^{M\times N}$ defined as in* (2.53)*. Further, let $\mathbb{S} = \mathbb{W}_R\,\mathbb{D}\,\mathbb{Z}_R^T = \mathbb{W}_R\,\mathbb{B}_R$ be the compact form of the SVD of $\mathbb{S}$, with $\mathbb{W}_R \in \mathbb{R}^{M\times R}$ and $\mathbb{Z}_R \in \mathbb{R}^{N\times R}$ defined as in* (2.58)*, $\mathbb{D} = \mathrm{diag}(\sigma_1, \dots, \sigma_R) \in \mathbb{R}^{R\times R}$, and $\mathbb{B}_R = \mathbb{D}\,\mathbb{Z}_R^T \in \mathbb{R}^{R\times N}$.*
Suppose that $\widehat{\mathbb{W}}_R \in \mathbb{R}^{M\times R}$ denotes a matrix with pairwise orthonormal columns $\widehat{\mathbf{w}}_r$, $r = 1, \dots, R$, and that the expansion of the columns of $\mathbb{S}$ in the basis $\{\widehat{\mathbf{w}}_n\}_{n=1}^N$ is given by

$$\mathbb{S} = \widehat{\mathbb{W}}_R\,\mathbb{C}_R\,,$$

with $\mathbb{C}_R \in \mathbb{R}^{R\times N}$, defined as

$$\left(\mathbb{C}_R\right)_{r,n} = \left(\widehat{\mathbf{w}}_r, \mathbf{s}_n\right)_{\mathbb{R}^M} \quad \textit{for } 1 \le r \le R,\ 1 \le n \le N.$$

Then for every $L \in \{1, \dots, R\}$ we have

$$\|\mathbb{S} - \mathbb{W}_L\,\mathbb{B}_L\|_F \le \left\|\mathbb{S} - \widehat{\mathbb{W}}_L\,\mathbb{C}_L\right\|_F. \tag{2.60}$$

Here, $\|\cdot\|_F$ denotes the Frobenius norm given by

$$\|\mathbb{A}\|_F = \sqrt{\sum_{m=1}^{M}\sum_{n=1}^{N}\left|A_{m,n}\right|^2} = \sqrt{tr\left(\mathbb{A}^T\mathbb{A}\right)} \quad \textit{for any } \mathbb{A}\in\mathbb{R}^{M\times N},$$

the matrix $\mathbb{W}_L$ (respectively, $\widehat{\mathbb{W}}_L$) denotes the first L columns of $\mathbb{W}$ (resp., $\widehat{\mathbb{W}}$), and $\mathbb{B}_L$ (resp., $\mathbb{C}_L$) denotes the first L rows of $\mathbb{B}$ (resp., $\mathbb{C}$) [58].

Regarding (2.60), let us note that

$$\begin{aligned}\left\|\mathbb{S}-\widehat{\mathbb{W}}_L\,\mathbb{C}_L\right\|_F^2 &= \sum_{m=1}^{M}\sum_{n=1}^{N}\left|\mathbb{S}_{m,n}-\sum_{l=1}^{L}\left(\widehat{\mathbb{W}}_L\right)_{m,l}\mathbb{C}_{l,n}\right|^2 = \sum_{n=1}^{N}\sum_{m=1}^{M}\left|(\mathbf{s}_n)_m-\sum_{l=1}^{L}\left(\mathbf{s}_n,\widehat{\mathbf{w}}_l\right)_{\mathbb{R}^M}(\mathbf{w}_l)_m\right| \\ &= \sum_{n=1}^{N}\left\|\mathbf{s}_n-\sum_{l=1}^{L}\left(\mathbf{s}_n,\widehat{\mathbf{w}}_l\right)_{\mathbb{R}^M}\mathbf{w}_l\right\|_{\mathbb{R}^M}^2 = .\end{aligned}$$

Then, according to (2.59),

$$\varepsilon(\widehat{\mathbf{w}}_1,\dots,\widehat{\mathbf{w}}_L) = \left\|\mathbb{S}-\widehat{\mathbb{W}}_L\,\mathbb{C}_L\right\|_F^2,$$

and, analogously,

$$\varepsilon(\mathbf{w}_1,\dots,\mathbf{w}_L) = \|\mathbb{S}-\mathbb{W}_L\,\mathbb{B}_L\|_F^2.$$

Hence, the optimality condition (2.60) implies

$$\varepsilon(\mathbf{w}_1,\dots,\mathbf{w}_L) \le \varepsilon(\widehat{\mathbf{w}}_1,\dots,\widehat{\mathbf{w}}_L)$$

for any set $\{\widehat{\mathbf{w}}_l\}_{l=1}^L$ of pairwise orthonormal vectors. Moreover, it can be shown that (see, e.g., [58])

$$\varepsilon(\mathbf{w}_1,\dots,\mathbf{w}_L) = \sum_{j=L+1}^{R}\sigma_j^2, \tag{2.61}$$

i.e., the error is given by the sum of the square of the discarded singular values.

The orthonormal basis $\{\mathbf{w}_1,\dots,\mathbf{w}_L\}$ is known as the *POD basis* of rank L. Returning to the POD-Galerkin RB method, we set $\boldsymbol{\psi}_l = \mathbf{w}_l$, for all $l = 1,\dots,L$, so that

$$\mathbb{V} = \left[\mathbf{w}_1\,\middle|\,\dots\,\middle|\,\mathbf{w}_L\right]^6.$$

Hence, in the reduced basis problem (2.47) 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

$$\psi_l(\boldsymbol{x}) = \sum_{m=1}^{M}\psi_l^{(m)}\,\phi_m(\boldsymbol{x}) = \sum_{m=1}^{M}(\mathbf{w}_l)_m\,\phi_m(\boldsymbol{x}) \quad \text{for } 1\le l\le L. \tag{2.62}$$

Let us point out that, in case of a scalar underlying differential equation, the POD basis functions $\{\psi_l\}_{l=1}^L$ are orthonormal on $V_h = X_h^r$ with respect to the following discrete scalar product $(\cdot,\cdot)_h$:

$$(\chi_h,\xi_h)_h = \sum_{i=1}^{M}\chi_h(\boldsymbol{N}_i)\,\xi_h(\boldsymbol{N}_i) = \sum_{i=1}^{M}\chi_h^{(i)}\,\xi_h^{(i)} = (\boldsymbol{\chi}_h,\boldsymbol{\xi}_h)_{\mathbb{R}^M}. \tag{2.63}$$

[6] Please observe that generally we do not decorate $\mathbb{V}$ with any subscript nor superscript reporting the dimension of the reduced basis it represents, unless not clear from the context.

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 $M \leq N$: directly solve the eigenvalue problems

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

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

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

Then, by (2.56) we have

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

Let us conclude by discussing the estimate (2.61) for the error committed by approximating the columns of the snapshot matrix $\mathbb{S}$ by means of the POD basis $\{\mathbf{w}_l\}_{l=1}^{M}$. First, note that it heavily relies on the magnitude of the first discarded singular value [29]. Luckily, in many situations the singular values show a graceful decay with the order, so that one can typically get accurate approximations by picking a few tens of basis vectors [8]. In particular, the minimum number of basis functions ensuring a POD error smaller than a desired tolerance δ corresponds with the smallest integer L satisfying the following inequality:

$$\frac{\sum_{l=1}^{L} \sigma_l^2}{\sum_{l=1}^{R} \sigma_l^2} > 1 - \delta. \tag{2.64}$$

In other terms, the energy retained by the first L POD modes must be greater than the fraction $1-\delta$ of the total energy of the snapshots [52]. This criterium, also known as *relative information content*, has been embodied into Algorithm 2.3 summarizing the POD method. Moreover, it also implies that

$$\frac{\|\mathbb{S} - \mathbb{W}_L \, \mathbb{B}_L\|_F}{\|\mathbb{S}\|_F} < \delta.$$

However, (2.61) and (2.64) only hold for the columns of $\mathbb{S}$, i.e., for the snapshots. While it can be readily generalized to any element in the snapshot manifold $\mathscr{M}_{\Xi_N}$ [7], we do not have any guarantee for all the other elements in the discrete solution manifold $\mathscr{M}_h$. Therefore, one may need a large number of shapshots, so ensuring that $\mathscr{M}_{\Xi_N}$ provides a good approximation of $\mathscr{M}_h$ and that the POD error can be bounded for a sufficiently large number of vectors. In our test cases, we will check the validity and reliability of the computed reduced basis *empirically*, by evaluating the projection error on a discrete and finite parameter test set $\Xi_{te} \subset \mathscr{P}$, with $\Xi_{te} \cap \Xi_N = \emptyset$ (see Chapter 3).

[7] Let $\mathbf{s} = \alpha_1 \, \mathbf{s}_1 + \ldots + \alpha_N \, \mathbf{s}_N \in \mathrm{Col}(\mathbb{V})$. Then:

$$\left\| \mathbf{s} - \sum_{l=1}^{L} \left(\mathbf{s}, \mathbf{w}_l\right)_{\mathbb{R}^M} \mathbf{w}_l \right\|_{\mathbb{R}^M}^2 = \left\| \sum_{n=1}^{N} \alpha_n \, \mathbf{s}_n - \sum_{n=1}^{N} \alpha_n \sum_{l=1}^{L} \left(\mathbf{s}_n, \mathbf{w}_l\right)_{\mathbb{R}^M} \mathbf{w}_l \right\|_{\mathbb{R}^M}^2$$

$$\leq \sum_{n=1}^{N} \alpha_n^2 \left\| \mathbf{s}_n - \sum_{l=1}^{L} \left(\mathbf{s}_n, \mathbf{w}_l\right)_{\mathbb{R}^M} \mathbf{w}_l \right\|_{\mathbb{R}^M}^2 \leq \left(\max_{1 \leq n \leq N} \alpha_n^2 \right) \sum_{j=L+1}^{R} \sigma_j^2.$$

Algorithm 2.3 The POD algorithm.

function $\mathbb{V} = \text{POD}(\mathbb{S}, \delta)$
  if $M \leq N$ **then**
    $\mathbb{M} = \mathbb{S}^T\mathbb{S}$
    for $i = 1, \dots, R$ **do**
      solve the eigenvalue problem $\mathbb{M}\,\mathbf{w}_i = \lambda_i\,\mathbf{w}_i$
    end for
  else
    $\mathbb{K} = \mathbb{S}\mathbb{S}^T$
    for $i = 1, \dots, R$ **do**
      solve the eigenvalue problem $\mathbb{K}\,\mathbf{z}_i = \lambda_i\,\mathbf{z}_i$
      $\mathbf{w}_i = \dfrac{1}{\sqrt{\lambda_i}}\mathbb{S}\,\mathbf{z}_i$
    end for
  end if
  find the minimum L satisfying (2.64)
  $\mathbb{V} = \left[\mathbf{w}_1 \,\middle|\, \dots \,\middle|\, \mathbf{w}_L\right]$
end function

2.5.2 Implementation: details and issues

The numerical procedure presented so far can be efficiently carried out within an offline-online framework [50]. 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. Then, given a new parameter value $\boldsymbol{\mu} \in \mathscr{P}$, the nonlinear reduced system (2.50) is solved *online* using, e.g., Newton's method, which entails the assembly and resolution of linear systems of the form (2.52). The main steps of the resulting POD-Galerkin (POD-G) RB method are summarized in Algorithm 2.4.

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

Algorithm 2.4 The offline and online stages for the POD-Galerkin (POD-G) RB method.

function $\left[\mathbb{V}, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n\right] = \text{PODG\_OFFLINE}(\mathscr{P}, \Omega_h, N, \delta_{\texttt{POD}}, \delta_{\texttt{NWT}}, K_{max})$
  solve the Laplace problems (2.27) and (2.28), yielding $\{\xi_i\}_{i=1}^n$ and $\{\pi_i\}_{i=1}^n$
  generate the parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$
  $\mathbf{u}_h(\boldsymbol{\mu}^{(i)}) = \text{NEWTON}(\boldsymbol{\mu}^{(i)}, \mathbf{u}_h^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\texttt{NWT}}, K_{max})$, for $i = 1, \dots, N$
  $\mathbb{S} = \left[\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \,\middle|\, \dots \,\middle|\, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})\right]$
  $\mathbb{V} = \text{POD}(\mathbb{S}, \delta_{\texttt{POD}})$
end function

function $\mathbf{u}_L(\boldsymbol{\mu}) = \text{PODG\_ONLINE}(\boldsymbol{\mu}, \Omega_h, \mathbb{V}, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\texttt{NWT}}, K_{max})$
  $\mathbf{u}_{\texttt{rb}}(\boldsymbol{\mu}) = \text{NEWTONRB}(\boldsymbol{\mu}, \Omega_h, \mathbb{V}, \mathbf{u}_{\texttt{rb}}^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\texttt{NWT}}, K_{max})$
  $\mathbf{u}_L(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\texttt{rb}}(\boldsymbol{\mu})$
end function

the operative definitions (2.50) and (2.51) 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 (2.52) 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) [48]. The basic idea is to recover an affine dependency on the parameter $\boldsymbol{\mu}$, approximating the reduced residual vector in the form

$$\mathbf{G}_{\mathtt{rb}}(\mathbf{w}_{\mathtt{rb}};\boldsymbol{\mu}) \approx \sum_{q=1}^{Q_g} \alpha_g^q(\mathbf{w}_{\mathtt{rb}};\boldsymbol{\mu})\, \mathbb{V}^T\mathbf{G}_h^q,$$

where $\{\mathbf{G}_h^q\}_{q=1}^{Q_g}$ represents a reduced basis for the space

$$\mathscr{M}_{\mathbf{G}} = \{\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu}) \,:\, \boldsymbol{\mu}\in\mathscr{P}\} \subset \mathbb{R}^M,$$

constructed *offline*, while $\alpha_g^q(\cdot;\cdot)$, $q=1,\dots,Q_g$, are *scalar* coefficients to be determined *online*. Similarly, for the Jacobian $\mathbb{J}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$ one can construct an affine approximation of the form

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{w}_{\mathtt{rb}};\boldsymbol{\mu}) \approx \sum_{q=1}^{Q_j} \alpha_j^q(\mathbf{w}_{\mathtt{rb}};\boldsymbol{\mu})\, \mathbb{V}^T\mathbb{J}_h^q\mathbb{V},$$

with $\{\mathbb{J}_h^q\}_{q=1}^{Q_j}$ a reduced basis for the space

$$\mathscr{M}_{\mathbb{J}} = \{\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu}) \,:\, \boldsymbol{\mu}\in\mathscr{P}\} \subset \mathbb{R}^{M\times M},$$

and $\alpha_j^q(\cdot;\cdot)$, $q=1,\dots,Q_j$, solution- and parameter-dependent scalar coefficients [52]. Notice that this approach guarantees a speed up with respect to the standard POD-Galerkin RB method by enabling the pre-computation of the large-sized terms $\mathbb{V}^T\mathbf{G}_h^q$ and $\mathbb{V}^T\mathbb{J}_h^q\mathbb{V}$, leaving to the online stage just the identification of the scalar coefficients $\alpha_g^q(\cdot;\cdot)$ and $\alpha_g^j(\cdot;\cdot)$.

In this work we shall not pursue this method. Rather, we propose an alternative, non-algebraic way to tackle online queries which bypass the assembly and resolution of the reduced system. We refer the reader to Section 2.6 for a complete description and further motivations. Before that, as an instructive example, let us briefly discuss the application of the (standard) POD-Galerkin method to the steady incompressible Navier-Stokes equations.

2.5.3 Application to the steady Navier-Stokes equations

Let $\{(\boldsymbol{v}_h(\boldsymbol{\mu}^{(n)}), p_h(\boldsymbol{\mu}^{(n)}))\}_{n=1}^N \subset V_h$ be an ensemble of velocity and pressure FE snapshots, corresponding to the parameter values $\{\boldsymbol{\mu}^{(1)},\dots,\boldsymbol{\mu}^{(N)}\}\subset\mathscr{P}$. Slightly distancing ourselves from the general workflow depicted in the previous sections, for the Navier-Stokes equations is convenient to construct two separate reduced spaces $V_{\mathtt{rb}}^{\boldsymbol{v}}$ and $V_{\mathtt{rb}}^{p}$ for the velocity and pressure fields, respectively [2, 8, 12, 52]; the advantages of this approach will be clearer later in the section. In light of this, let

$$\mathbb{S}_{\boldsymbol{v}} = \left[\mathbf{v}_h(\boldsymbol{\mu}^{(1)}) \,\middle|\, \dots \,\middle|\, \mathbf{v}_h(\boldsymbol{\mu}^{(N)})\right] \in \mathbb{R}^{2M_n\times N} \quad \text{and} \quad \mathbb{S}_p = \left[\mathbf{p}_h(\boldsymbol{\mu}^{(1)}) \,\middle|\, \dots \,\middle|\, \mathbf{p}_h(\boldsymbol{\mu}^{(N)})\right] \in \mathbb{R}^{M_v\times N}$$

be the velocity and pressure snapshots matrices, and consider their singular values decomposition:

$$\mathbb{S}_{\boldsymbol{v}} = \mathbb{W}_{\boldsymbol{v}} \Sigma_{\boldsymbol{v}} \mathbb{Z}_{\boldsymbol{v}}^T \quad \text{and} \quad \mathbb{S}_p = \mathbb{W}_p \Sigma_p \mathbb{Z}_p^T,$$

where $\mathbb{W}_{\boldsymbol{v}} \in \mathbb{R}^{2M_n \times 2M_n}$, $\mathbb{Z}_{\boldsymbol{v}} \in \mathbb{R}^{N\times N}$, $\mathbb{W}_p \in \mathbb{R}^{M_v \times M_v}$ and $\mathbb{Z}_p \in \mathbb{R}^{N\times N}$ are orthogonal matrices, while $\Sigma_{\boldsymbol{v}} \in \mathbb{R}^{2M_n \times N}$ and $\Sigma_p \in \mathbb{R}^{M_v \times N}$ are diagonal matrices (see Section 2.5.1). We then denote by $\mathbb{V}_{\boldsymbol{v}} \in \mathbb{R}^{2M_n \times L_{\boldsymbol{v}}}$ and $\mathbb{V}_p \in \mathbb{R}^{M_v \times L_p}$ the matrices retaining, respectively, the first $L_{\boldsymbol{v}}$ columns of $\mathbb{W}_{\boldsymbol{v}}$ and the first L_p columns of $\mathbb{W}_p$, namely,

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

with $\{\boldsymbol{\psi}_i^{\boldsymbol{v}}\}_{i=1}^{2M_n}$ and $\{\boldsymbol{\psi}_i^{p}\}_{i=1}^{M_v}$ the left singular vectors of $\mathbb{S}_{\boldsymbol{v}}$ and $\mathbb{S}_p$, respectively. Hence, for any $\boldsymbol{\mu} \in \mathscr{P}$ we seek reduced-order approximations $\mathbf{v}_L(\boldsymbol{\mu}) \in \mathbb{R}^{2M_n}$ and $\mathbf{p}_L(\boldsymbol{\mu}) \in \mathbb{R}^{M_v}$ for the state variables in the form

$$\mathbf{v}_L(\boldsymbol{\mu}) = \mathbb{V}_{\boldsymbol{v}} \mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}) \approx \mathbf{v}_h(\boldsymbol{\mu}) \quad \text{and} \quad \mathbf{p}_L(\boldsymbol{\mu}) = \mathbb{V}_p \mathbf{p}_{\mathtt{rb}}(\boldsymbol{\mu}) \approx \mathbf{p}_h(\boldsymbol{\mu}),$$

with $\mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^{L_{\boldsymbol{v}}}$ and $\mathbf{p}_{\mathtt{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^{L_p}$. Inserting the expressions for $\mathbf{v}_L(\boldsymbol{\mu})$ and $\mathbf{p}_L(\boldsymbol{\mu})$ into the FE system (2.44) and enforcing the orthogonality with the columns of $\mathbb{V}_{\boldsymbol{v}}$ and $\mathbb{V}_p$, yields the following reduced-order nonlinear system for the $L = L_{\boldsymbol{v}} + L_p$ RB coefficients gathered in $\mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu})$ and $\mathbf{p}_{\mathtt{rb}}(\boldsymbol{\mu})$:

$$\begin{bmatrix} \mathbb{V}_{\boldsymbol{v}}^T & 0 \\ 0 & \mathbb{V}_p^T \end{bmatrix} \begin{bmatrix} \mathbb{A}(\boldsymbol{\mu}) + \mathbb{C}\big(\mathbb{V}_{\boldsymbol{v}} \mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}\big) & \mathbb{B}(\boldsymbol{\mu})^T \\ \mathbb{B}(\boldsymbol{\mu}) & 0 \end{bmatrix} \begin{bmatrix} \mathbb{V}_{\boldsymbol{v}} \mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}) \\ \mathbb{V}_p \mathbf{p}_{\mathtt{rb}}(\boldsymbol{\mu}) \end{bmatrix} = \begin{bmatrix} \mathbb{V}_{\boldsymbol{v}}^T & 0 \\ 0 & \mathbb{V}_p^T \end{bmatrix} \begin{bmatrix} \mathbf{f}_1(\boldsymbol{\mu}) \\ \mathbf{f}_2(\boldsymbol{\mu}) \end{bmatrix}. \tag{2.65}$$

Developing the calculations and defining

$$\begin{aligned} &\mathbb{A}_L(\boldsymbol{\mu}) = \mathbb{V}_{\boldsymbol{v}}^T \mathbb{A}(\boldsymbol{\mu}) \mathbb{V}_{\boldsymbol{v}} \in \mathbb{R}^{L_{\boldsymbol{v}} \times L_{\boldsymbol{v}}}, \quad \mathbb{C}_L\big(\mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}\big) = \mathbb{V}_{\boldsymbol{v}}^T \mathbb{C}(V_{\boldsymbol{v}} \mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) \mathbb{V}_{\boldsymbol{v}} \in \mathbb{R}^{L_{\boldsymbol{v}} \times L_{\boldsymbol{v}}}, \\ &\mathbb{B}_L(\boldsymbol{\mu}) = \mathbb{V}_p^T \mathbb{B}(\boldsymbol{\mu}) \mathbb{V}_{\boldsymbol{v}} \in \mathbb{R}^{L_p \times L_{\boldsymbol{v}}}, \quad \mathbf{h}_1 = \mathbb{V}_{\boldsymbol{v}}^T \mathbf{f}_1(\boldsymbol{\mu}) \in \mathbb{R}^{L_{\boldsymbol{v}}}, \quad \mathbf{h}_2 = \mathbb{V}_p^T \mathbf{f}_2(\boldsymbol{\mu}) \in \mathbb{R}^{L_p}, \end{aligned}$$

the system (2.65) can be expressed in compact form as

$$\begin{bmatrix} \mathbb{A}_L(\boldsymbol{\mu}) + \mathbb{C}_L\big(\mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}\big) & \mathbb{B}_L(\boldsymbol{\mu})^T \\ \mathbb{B}_L(\boldsymbol{\mu}) & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}) \\ \mathbf{p}_{\mathtt{rb}}(\boldsymbol{\mu}) \end{bmatrix} = \begin{bmatrix} \mathbf{h}_1(\boldsymbol{\mu}) \\ \mathbf{h}_2(\boldsymbol{\mu}) \end{bmatrix}.$$

Let us point out that the reduced system (2.65) can be recast in the canonical form (2.50) upon setting

$$\mathbb{V} = \begin{bmatrix} \mathbb{V}_{\boldsymbol{v}} & 0 \\ 0 & \mathbb{V}_P \end{bmatrix} \in \mathbb{R}^{M \times L}.$$

Although in Section 2.4.2 the underpinning finite element solver has been designed to fulfill the inf-sup condition (2.34), thus ensuring the stability of the scheme, when dealing with incompressible flows, this property may not be automatically inherited by the reduced-order solver, which as a result may show severe instabilities [8]. Indeed, one has to carefully chosen the reduced velocity and pressure spaces so to meet an equivalent of the inf-sup condition (2.67) at the reduced level. To this end, in our simulations we resort to a *supremizer enrichment* of the reduced velocity space $X_{\mathtt{rb}}$, as proposed in [2], to which we refer the readers which are interested in the theoretical rational behind this approach.

In the following, we rather provide some insights on the algorithmic and implementative aspects.

As for the full-order scheme, to retain the stability of the POD-Galerkin method we have to ensure that $X_{\mathtt{rb}}$ is somehow *richer* than the reduced search space $Q_{\mathtt{rb}}$ for the pressure. For this purpose, with each pressure snapshot $\mathbf{p}_h(\boldsymbol{\mu}^{(n)})$, $n = 1, \dots N$, we associate the *supremizer* solution $\mathbf{s}_h(\boldsymbol{\mu}^{(n)}) \in \mathbb{R}^{2M_n}$, defined as the solution of the linear system

$$\mathbb{M}_{\boldsymbol{v}}\, \mathbf{s}(\boldsymbol{\mu}^{(n)}) = \mathbb{B}(\boldsymbol{\mu}^{(n)})^T \mathbf{p}_h(\boldsymbol{\mu}^{(n)}), \quad \text{for } n = 1, \dots, N. \tag{2.66}$$

where $\mathbb{B} = \mathbb{B}(\boldsymbol{\mu}) \in \mathbb{R}^{M_v \times 2M_n}$ has been introduced in Eq. (2.4.2), while $\mathbb{M}_{\boldsymbol{v}} \in \mathbb{R}^{2M_n \times 2M_n}$ denotes the mass matrix in the $\left[H^1(\Omega)\right]^2$ scalar product for the velocity FE basis functions, and it is defined by

$$\left(\mathbb{M}_{\boldsymbol{v}}\right)_{i,j} = \left(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j\right)_{H^1(\Omega)} = \sum_{K \in \Omega_h} \int_K \left[\boldsymbol{\phi}_i \cdot \boldsymbol{\phi}_j + \nabla \boldsymbol{\phi}_i : \nabla \boldsymbol{\phi}_j\right] dK, \quad \text{for } i, j = 1, \dots, 2M_n.$$

Letting $\mathbb{S}_{\boldsymbol{s}} = \left[\mathbf{s}(\boldsymbol{\mu}^{(1)}) \,\middle|\, \dots \,\middle|\, \mathbf{s}(\boldsymbol{\mu}^{(N)})\right] \in \mathbb{R}^{2M_n \times N}$ collect the coefficients for the expansion of the supremizer solutions in the Langrangian FE basis, we generate a POD basis for the supremizer space by performing the SVD of $\mathbb{S}_{\boldsymbol{s}}$, and we then enrich the reduced basis for the velocity field by means of the first $L_{\boldsymbol{s}}$ left singular vectors $\{\boldsymbol{\psi}_i^{\boldsymbol{s}}\}_{i=1}^{L_{\boldsymbol{s}}}$, yielding an enriched reduced velocity space $\bar{X}_{\mathtt{rb}}$. In other words, setting

$$\mathbb{V}_{\boldsymbol{s}} = \left[\boldsymbol{\psi}_1^{\boldsymbol{s}} \,\middle|\, \dots \,\middle|\, \boldsymbol{\psi}_{L_{\boldsymbol{s}}}^{\boldsymbol{s}}\right] \in \mathbb{R}^{2M_n \times L_{\boldsymbol{s}}},$$

a reduced-order approximation for the velocity field is now sought in the form

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

where

$$\overline{\mathbb{V}}_{\boldsymbol{v}} = [\mathbb{V}_{\boldsymbol{v}}\ \mathbb{V}_{\boldsymbol{s}}] \in \mathbb{R}^{2M_n \times (L_{\boldsymbol{v}} + L_{\boldsymbol{s}})} \quad \text{and} \quad \bar{\mathbf{v}}_{\mathtt{rb}}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{v}_{\mathtt{rb}}(\boldsymbol{\mu}) \\ \mathbf{s}_{\mathtt{rb}}(\boldsymbol{\mu}) \end{bmatrix} \in \mathbb{R}^{L_{\boldsymbol{v}} + L_{\boldsymbol{s}}}.$$

Replacing $\mathbb{V}_{\boldsymbol{v}}$ with $\overline{\mathbb{V}}_{\boldsymbol{v}}$ and $\mathbf{v}_{\mathtt{rb}}$ with $\bar{\mathbf{v}}_{\mathtt{rb}}$ into (2.65), we get the reduced nonlinear system for $\left(\bar{\mathbf{v}}_{\mathtt{rb}}(\boldsymbol{\mu}), \mathbf{p}_{\mathtt{rb}}(\boldsymbol{\mu})\right)$, consisting of $L_{\boldsymbol{s}}$ additional equations. The stability of the resulting reduced-order scheme is ensured by the following proposition [2].

Proposition 2.1. *Let $\bar{X}_{\mathtt{rb}}$ and $Q_{\mathtt{rb}}$ be endowed with the scalar products $(\cdot,\cdot)_{\mathtt{rb}}$ and $(\cdot,\cdot)_{\mathtt{rb}}$, respectively, defined as*

$$(\bar{\boldsymbol{w}}_L, \bar{\boldsymbol{z}}_L)_{\mathtt{rb}} = \bar{\mathbf{w}}_{\mathtt{rb}}^T\, \mathbb{M}_{\boldsymbol{v}}^{\mathtt{rb}}\, \bar{\mathbf{z}}_{\mathtt{rb}} \quad \textit{and} \quad (q_L, r_L) = \mathbf{q}_{\mathtt{rb}}^T\, \mathbb{X}_p^{\mathtt{rb}}\, \mathbf{r}_{\mathtt{rb}},$$

where

$$\mathbb{M}_{\boldsymbol{v}}^{\mathtt{rb}} = \begin{bmatrix} \mathbb{V}_{\boldsymbol{v}}^T \mathbb{M}_{\boldsymbol{v}} \mathbb{V}_{\boldsymbol{v}} & \mathbb{V}_{\boldsymbol{v}}^T \mathbb{M}_{\boldsymbol{v}} \mathbb{V}_{\boldsymbol{s}} \\ \mathbb{V}_{\boldsymbol{s}}^T \mathbb{M}_{\boldsymbol{v}} \mathbb{V}_{\boldsymbol{v}} & \mathbb{V}_{\boldsymbol{s}}^T \mathbb{M}_{\boldsymbol{v}} \mathbb{V}_{\boldsymbol{s}} \end{bmatrix} \in \mathbb{R}^{(L_{\boldsymbol{v}} + L_{\boldsymbol{s}}) \times (L_{\boldsymbol{v}} + L_{\boldsymbol{s}})} \quad \textit{and} \quad \mathbb{M}_p^{\mathtt{rb}} = \mathbb{V}_p^T\, \mathbb{M}_p\, \mathbb{V}_p \in \mathbb{R}^{L_p \times L_p},$$

with $\mathbb{M}_{\boldsymbol{v}}$ and $\mathbb{M}_p$ being the mass matrices for the velocity and pressure (full-order) spaces, respectively. Assuming that the full-order model satisfies the inf-sup stability condition (2.43)*, the POD-Galerkin ROM with supremizer enrichment of the velocity space is inf-sup stable, that is*

$$\beta_L(\boldsymbol{\mu}) = \inf_{q_L \in Q_{\mathtt{rb}}} \sup_{\boldsymbol{w}_L \in \bar{X}_{\mathtt{rb}}} \frac{b_h(q_L, \boldsymbol{w}_L; \boldsymbol{\mu})}{\|\boldsymbol{w}_L\|_{\mathtt{rb}} \left\|q_L\right\|_{\mathtt{rb}}} \geq \beta_{0\,L} \geq \beta_0 > 0 \quad \forall\, \boldsymbol{\mu} \in \mathscr{P}. \tag{2.67}$$

2.6 A POD-based RB method using neural networks

Although intuitive, as resulting from a straightforward projection procedure, a refined analysis may reveal a few drawbacks and open questions for the standard POD-Galerkin reduced basis technique, particularly for nonlinear differential problems featuring a nonaffine dependence on the parameters. In those cases, we have already discussed in Section 2.5.2 about the incapability of the method to completely decouple the online stage from the underlying high-fidelity, expensive discrete model, thus preventing a substantial speed up with respect to standard discretization methods. Albeit one can (at least partially) overcome this issue upon resorting to suitable interpolation techniques as DEIM or MDEIM, 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].

Furthermore, recall that, at the algebraic level, we seek an approximated solution of the form

$$\mathbf{u}_L = \mathbb{V}\,\mathbf{u}_{\mathtt{rb}}\,,$$

i.e., belonging to the column space $\mathrm{Col}(\mathbb{V})$ of $\mathbb{V}$, as the solution of the reduced system

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

We have seen that the above system encodes the projection of the original variational model onto the reduced space $V_{\mathtt{rb}}$. For our purposes, it is worth remarking that there exists a one-to-one correspondence between $V_{\mathtt{rb}}$ and $\mathrm{Col}(\mathbb{V})$. 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. (2.62) follows:

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

In particular, this implies that the projection of any $v_h \in V_h$ onto $V_{\mathtt{rb}}$ in the discrete scalar product $(\cdot,\cdot)_h$ (see (2.63)) algebraically corresponds to the projection of $\mathbf{v}_h$ onto $\mathrm{Col}(\mathbb{V})$ in the Euclidean scalar product, given by

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

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

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

One can reasonably expect that the solution to the reduced system approaches or even coincides with the projection of the (unknown) high-fidelity solution $\mathbf{u}_h$ onto $\mathrm{Col}(\mathbb{V})$, i.e.,

$$\mathbf{u}_L \approx \mathbb{P}\,\mathbf{u}_h = \mathbb{V}\mathbb{V}^T\mathbf{u}_h\,, \tag{2.68a}$$

or equivalently

$$\mathbf{u}_{\mathtt{rb}} \approx \mathbb{V}^T \mathbf{u}_h \,. \tag{2.68b}$$

To observe this, let us briefly consider the following boundary value problem for the nonlinear two-dimensional Poisson equation:

$$\begin{cases} -\widetilde{\nabla} \cdot \left(\left(1 + \widetilde{u}(\boldsymbol{\mu})^2 \right) \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \right) = \widetilde{s}(\widetilde{x}, \widetilde{y}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}) \,, \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{\sigma}_y \sin(\widetilde{\sigma}_x) \cos(\widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu}) \,. \end{cases} \tag{2.69}$$

where $\widetilde{\Omega}(\boldsymbol{\mu})$ [8] is the quadrilateral domain shown in Fig. 2.3, with $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \left[\pi/4, 3\pi/4 \right] \times \left[0, \pi/2 \right]$, and the source term $\widetilde{s}$ has been chosen so that the exact solution to the problem is given by $\widetilde{u}_{ex} = \widetilde{y} \sin(\widetilde{x}) \cos(\widetilde{y})$. Assume the POD modes have been constructed relying on $N = 100$ parameter samples picked via the Latin Hypercube Sampling (LHS) [32], with the corresponding snapshots computed over a triangular mesh consisting of 1921 vertices. On the left in Fig. 2.4, we compare the decay with L (i.e., the dimension of the reduced basis) of the relative errors between the high-fidelity solution and either its projection onto $V_{\mathtt{rb}}$ (red squares) or the POD-Galerkin RB approximation (blue circles). Given $\boldsymbol{\mu} \in \mathscr{P}$, the errors are respectively defined as

$$\varepsilon_{\mathbb{V}}(L, \boldsymbol{\mu}) = \frac{\left\| \mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) \right\|}{\left\| \mathbf{u}_h(\boldsymbol{\mu}) \right\|} \quad \text{and} \quad \varepsilon_{\mathtt{PODG}}(L, \boldsymbol{\mu}) = \frac{\left\| \mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_L(\boldsymbol{\mu}) \right\|}{\left\| \mathbf{u}_h(\boldsymbol{\mu}) \right\|} \,.$$

In particular, the plot reports the average values $\bar{\varepsilon}_{\mathbb{V}}(L)$ and $\bar{\varepsilon}_{\mathtt{PODG}}(L)$ of $\varepsilon_{\mathbb{V}}(L, \boldsymbol{\mu})$ and $\varepsilon_{\mathtt{PODG}}(L, \boldsymbol{\mu})$, respectively, evaluated over a test parameter set $\Xi_{te} \subset \mathscr{P}$ consisting of $N_{te} = 50$ samples, with $\Xi_{te} \cap \Xi_N = \emptyset$. We observe that the Galerkin-POD scheme shows an exponential convergence to the high-fidelity solution, attaining a relative error of 0.2% for $L = 35$, which turns out to be rather close to theoretical lower-bound provided by the projection error. However, even if the solver successfully converges to the solution, resorting to a reduced-order strategy only seldomly enables any (actually small) computational saving with respect to the finite element method, as shown by the plot on the right-hand side in Fig. 2.4, reporting the online run times on each test sample for both the full-order and the reduced-order scheme [9]. As already pointed out, this has to be ascribed to the nonaffine parametric dependence featured by the problem.

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, we have noted above that 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

$$u_h^{\mathbb{V}}(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{j=1}^{M} \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) \right)_j \phi_j(\boldsymbol{x}) = \sum_{i=1}^{L} \left(\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) \right)_i \psi_i(\boldsymbol{x}) \,. \tag{2.70}$$

[8] Observe that here only geometrical parameters are involved, i.e., $\boldsymbol{\mu} \equiv \boldsymbol{\mu}_g$.

[9] In this respect, we mention that the results presented throughout this thesis have been obtained via a MATLAB® implementation, with all the simulations carried out on a Lenovo laptop equipped with a CPU Intel Core i7 @ 2.50 GHz.

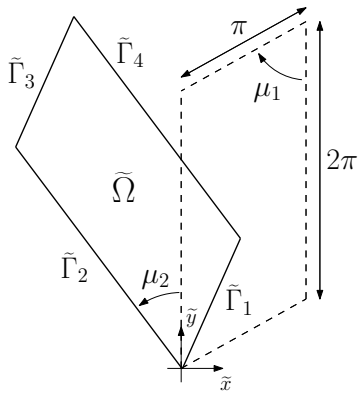


Figure 2.3. The parametrized computational domain (solid line) for the Poisson problem (2.69).

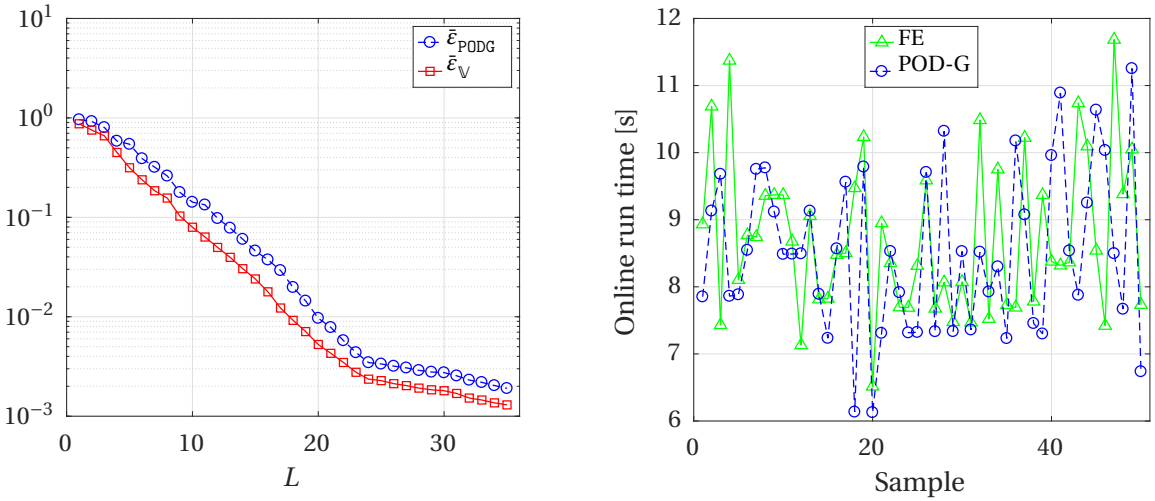


Figure 2.4. Left: average relative error between the FE solution for problem (2.69) and either its projection onto the reduced space (red squares) or the POD-Galerkin RB solution (blue circles); the errors have been evaluated on a test parameter set $\Xi_{te} \subset \mathscr{P}$ consisting of $N_{te} = 50$ randomly picked values. Right: comparison between the online run times for the FE scheme and the POD-Galerkin method on Ξ_{te}, with $L = 35$ basis functions included in the reduced model.

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

$$\begin{aligned} \boldsymbol{\pi} : \mathscr{P} \subset \mathbb{R}^P &\to \mathbb{R}^L \\ \boldsymbol{\mu} &\mapsto \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}), \end{aligned} \tag{2.71}$$

mapping 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,

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

It is worth pointing out 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 $\boldsymbol{\pi}$; we shall appreciate the consequence of these facts in the next section.

In the literature, different approaches for the *interpolation* of (2.71) 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 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 Chapter 1, any neural network is tailored for 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

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

According to the notation and nomenclature introduced in the previous chapter, for $i = 1, \dots, N_{tr}$, $\mathbf{p}_i = \boldsymbol{\mu}^{(i)} \in \mathbb{R}^P$ represents the *input pattern* and $\mathbf{t}_i = \mathbb{V}^T \mathbf{u}_h\left(\boldsymbol{\mu}^{(i)}\right) \in \mathbb{R}^L$ the associated *teaching input*; together, they constitute a *training pattern*. In this respect, note that the teaching inputs $\mathbb{V}^T \mathbf{u}_h\left(\boldsymbol{\mu}^{(i)}\right)$, $i = 1, \dots, N_{tr}$, 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 (conversely to the reduced solver, as already appreciated before). 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 2.6. 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; we refer the reader to the following chapter for a numerical validation of this assertion.

However, let us point 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 discussed in Chapter 1 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 Algorithm 2.5.

While the Cybenko's theorems (see (i) and (ii) in Section 1.2.2) allows us 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) We can now limit ourselves to network configurations including a number of neurons non-smaller than the current optimal network. 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, which has proved to provide a good compromise between randomness and even covarage of the parameter domain; an example is offered in Fig. 2.5 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.

Algorithm 2.5 Selection of an optimal network configuration.

function $\left(\mathscr{N}^{opt}, \mathscr{V}^{opt}, \mathbf{w}^{opt}\right) =$
  PODNN_TRAINING($\mathbb{V}$, ε, N_{tr}^{max}, H^{max}, N_{te}, d, K_{ms}, δ, T, K_{ea})
  set $\Delta N = 100$, $H_0 = 5$, $\Delta H = 5$, $N_{va} = \lceil 0.3 N_{tr}^{max} \rceil$ ▷ Adjustable parameters
  randomly drawn validation and test samples $\Xi_{va} = \left\{\boldsymbol{\mu}_{va}^{(i)}\right\}_{i=1}^{N_{va}}$ and $\Xi_{te} = \left\{\boldsymbol{\mu}_{te}^{(i)}\right\}_{i=1}^{N_{te}}$
  compute $\left\{\mathbf{u}_h\left(\boldsymbol{\mu}_{va}^{(i)}\right)\right\}_{i=1}^{N_{va}}$ and $\left\{\mathbf{u}_h\left(\boldsymbol{\mu}_{te}^{(i)}\right)\right\}_{i=1}^{N_{te}}$
  set $P_{va} = \left\{\left(\boldsymbol{\mu}_{va}^{(i)}, \mathbb{V}^T \mathbf{u}_h\left(\boldsymbol{\mu}_{va}^{(i)}\right)\right)\right\}_{i=1}^{N_{va}}$ and $P_{te} = \left\{\left(\boldsymbol{\mu}_{te}^{(i)}, \mathbb{V}^T \mathbf{u}_h\left(\boldsymbol{\mu}_{te}^{(i)}\right)\right)\right\}_{i=1}^{N_{te}}$
  evaluate projection error $E_{\mathtt{POD}} = {}^1/_{N_{te}} \sum_{i=1}^{N_{te}} \left\| \mathbf{u}_h\left(\boldsymbol{\mu}_{te}^{(i)}\right) - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\left(\boldsymbol{\mu}_{te}^{(i)}\right) \right\|$
  $P_{tr} = \emptyset$, $N_{tr} = 0$, $H^{opt} = H_0$
  do
    $N_{tr} \leftarrow N_{tr} + \Delta N$
    generate training samples $\Xi_{tr} = \left\{\boldsymbol{\mu}_{tr}^{(i)}\right\}_{i=1}^{N_{tr}}$ via Latin Hypercube Sampling (LHS)
    compute snapshots $\left\{\mathbf{u}_h\left(\boldsymbol{\mu}_{tr}^{(i)}\right)\right\}_{i=1}^{N_{tr}}$
    extend training data set: $P_{tr} \leftarrow P_{tr} \cup \left\{\left(\boldsymbol{\mu}_{tr}^{(i)}, \mathbb{V}^T \mathbf{u}_h\left(\boldsymbol{\mu}_{tr}^{(i)}\right)\right)\right\}_{i=1}^{N_{tr}}$
    $E_{\mathtt{NN}}^{opt} = \infty$, $H = H^{opt}$
    while $H \leq H^{max}$ and $E_{te} < E_{te}^{old}$ **do**
      design a three-layers perceptron $\left(\mathscr{N}, \mathscr{V}\right)$ with H neurons per hidden layer
      $\left[\mathbf{w}, E_{\mathtt{NN}}\right] =$ TRAINING($(\mathscr{N}, \mathscr{V})$, P_{tr}, P_{va}, P_{te}, d, K_{ms}, δ, T, K_{ea})
      if $E_{\mathtt{NN}} < E_{\mathtt{NN}}^{opt}$ **then**
        $E_{\mathtt{NN}}^{opt} = E_{\mathtt{NN}}$, $\mathscr{N}^{opt} = \mathscr{N}$, $\mathscr{V}^{opt} = \mathscr{V}$, $\mathbf{w}^{opt} = \mathbf{w}$, $H^{opt} = H$
        $H \leftarrow H + \Delta H$
      else ▷ Too many neurons: the network overfits training data
        break
      end if
    end while
  while $N_{tr} \leq N_{tr}^{max}$ and $\left| E_{\mathtt{POD}} - E_{\mathtt{NN}}^{opt} \right| > \varepsilon$
end function

As described in Section 1.2.3, 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) (1.10) 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):

$$\begin{aligned} MSE\big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}, \mathbb{V}^T\mathbf{u}_h\big) \propto \big\|\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big\|_{\mathbb{R}^L}^2 &= \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big)^T\big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big) \\ &= \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big)^T \underbrace{\mathbb{V}^T\mathbb{V}}_{\mathbb{I}_L \in \mathbb{R}^{L\times L}} \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big) \\ &= \big(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big)^T\big(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big) \\ &= \big\|\mathbf{u}_L^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big\|_{\mathbb{R}^M}^2 = \big\|u_L^{\mathtt{NN}} - u_h^{\mathbb{V}}\big\|_h^2, \end{aligned} \tag{2.72}$$

where

$$u_L^{\mathtt{NN}}(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{i=1}^{L} \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})\big)_i\, \psi_i(\boldsymbol{x}) \in V_{\mathtt{rb}}. \tag{2.73}$$

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, generalized cross validation) aiming at preventing the network to *overfit* the training data; see Section 1.2.4 for further details.

While we shall numerically show the actual effectiveness and efficiency of the POD-NN method in the upcoming chapter, in the following concluding section 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.

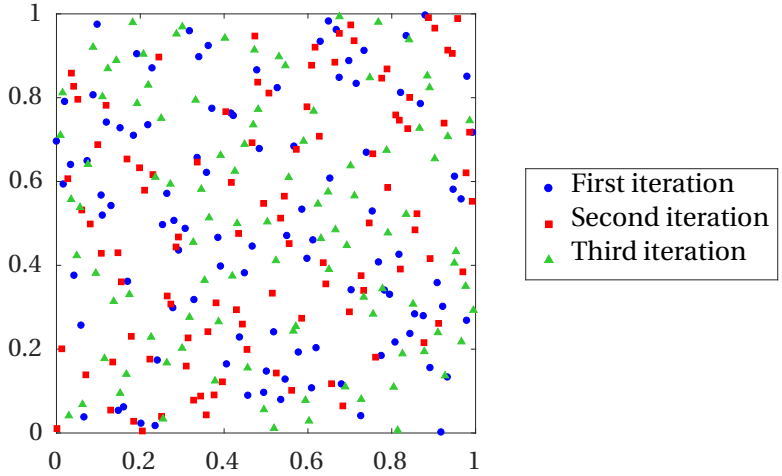


Figure 2.5. Three point sets randomly generate through the latin hypercube sampling. Observe the good coverage provided by the union of the sets, with only a few overlapping points.

Algorithm 2.6 The offline and online stages for the POD-NN RB method.

function $\left[\mathbb{V}, \mathscr{N}_{\mathtt{rb}}, \mathscr{V}_{\mathtt{rb}}, \boldsymbol{w}_{\mathtt{rb}}\right]$ = PODNN_OFFLINE($\mathscr{P}$, Ω_h, N, $\delta_{\mathtt{NWT}}$, K_{max}, $\delta_{\mathtt{POD}}$, $\delta_{\mathtt{NN}}$, ε, N_{tr}^{max}, H^{max}, N_{te}, d, K_{ms}, T, K_{ea})
  solve the Laplace problems (2.27) and (2.28), yielding $\{\xi_i\}_{i=1}^n$ and $\{\pi_i\}_{i=1}^n$
  generate the parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$
  $\mathbf{u}_h(\boldsymbol{\mu}^{(i)})$ = NEWTON($\boldsymbol{\mu}^{(i)}$, $\mathbf{u}_h^0$, $\{\xi_i\}_{i=1}^n$, $\{\pi_i\}_{i=1}^n$, $\delta_{\mathtt{NWT}}$, K_{max}), for $i = 1, \dots, N$
  $\mathbb{S} = \left[\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \,\middle|\, \dots \,\middle|\, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})\right]$
  $\mathbb{V}$ = POD($\mathbb{S}$, $\delta_{\mathtt{POD}}$)
  $\left(\mathscr{N}_{\mathtt{rb}}, \mathscr{V}_{\mathtt{rb}}, \mathbf{w}_{\mathtt{rb}}\right)$ = PODNN_TRAINING($\mathbb{V}$, ε, N_{tr}^{max}, H^{max}, N_{te}, d, K_{ms}, $\delta_{\mathtt{NN}}$, T, K_{ea})
end function

function $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu})$ = PODNN_ONLINE($\boldsymbol{\mu}$, $\mathbb{V}$, $\mathscr{N}_{\mathtt{rb}}$, $\mathscr{V}_{\mathtt{rb}}$, $\mathbf{w}_{\mathtt{rb}}$)
  evaluate the output $\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$ of the network $\left(\mathscr{N}_{\mathtt{rb}}, \mathscr{V}_{\mathtt{rb}}, \mathbf{w}_{\mathtt{rb}}\right)$ for the input vector $\boldsymbol{\mu}$
  $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu}) = \mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$
end function

2.6.1 An a priori error analysis

For the sake of convenience, we directly consider the variational problem (2.13) stated over the reference domain Ω. As usual, let $u(\boldsymbol{\mu}) \in V \subseteq H^1(\Omega)$ be the exact weak solution, $u_h(\boldsymbol{\mu}) \in V_h$ the high-fidelity discrete solution (obtained, e.g., through the finite element method) with $u_h^{\mathbb{V}}(\boldsymbol{\mu}) \in V_{\mathtt{rb}}$ its projection onto $V_{\mathtt{rb}}$, and $u_L^{\mathtt{NN}}(\boldsymbol{\mu}) \in V_{\mathtt{rb}}$ the reduced solution provided by the POD-NN method, defined in Eq. (2.73). Omitting the dependence on the parameter $\boldsymbol{\mu}$, a straightforward application of the triangular inequality yields the following upper bound for the $L^2(\Omega)$-norm of the error committed by POD-NN RB method:

$$\left\| u - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} \leq \| u - u_h \|_{L^2(\Omega)} + \left\| u_h - u_h^{\mathbb{V}} \right\|_{L^2(\Omega)} + \left\| u_h^{\mathbb{V}} - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)}. \tag{2.74}$$

Let us analyze the three terms appearing on the right-hand side of (2.74). The former quantifies the discrepancy between the exact solution u and the discrete approximation u_h provided by the full-order solver. Throughout, we have assumed that u_h can be driven as close as desired to u in the V-norm; for instance, the FE solution can be improved either by refining the underlying mesh Ω_h, or increasing the order of the interpolating polynomials, or both. Therefore:

$$\| u - u_h \|_{L^2(\Omega)} \leq \| u - u_h \|_V \leq \delta_{\mathtt{HF}}, \tag{2.75}$$

with $\delta_{\mathtt{HF}} > 0$ being a given tolerance. Then, the term $\left\| u_h - u_h^{\mathbb{V}} \right\|_{L^2(\Omega)}$ measures the distance between the truth solution and the reduced space $V_{\mathtt{rb}}$. From the definitions (2.35) and (2.70), it follows that:

$$\begin{aligned}
\left\| u_h - u_h^{\mathbb{V}} \right\|_{L^2(\Omega)}^2 &= \int_\Omega \left| u_h - u_h^{\mathbb{V}} \right|^2 d\Omega = \int_\Omega \left| \sum_{i=1}^M \left(\mathbf{u}_h\right)_i \phi_i - \sum_{i=1}^M \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h\right)_i \phi_i \right|^2 d\Omega \\
&= \sum_{i=1}^M \sum_{j=1}^M \left(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\right)_i \left(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\right)_j \underbrace{\int_\Omega \phi_i \phi_j \, d\Omega}_{\mathbb{M}_{i,j}} \\
&= \left(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\right)^T \mathbb{M} \left(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\right).
\end{aligned}$$

where $\mathbb{M} \in \mathbb{R}^{M\times M}$ denotes the mass matrix. Exploiting the symmetry and positive definiteness of $\mathbb{M}$ (following from the symmetry and positiveness of the canonical scalar product $(\cdot,\cdot)_{L^2(\Omega)}$ of $L^2(\Omega)$), we further get:

$$\left\| u_h - u_h^{\mathbb{V}} \right\|_{L^2(\Omega)}^2 = \left\| \mathbb{M}^{1/2}\left(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T\mathbf{u}_h\right) \right\|_{\mathbb{R}^M}^2 \leq \|\mathbb{M}\|_2 \left\| \mathbf{u}_h - \mathbb{V}\mathbb{V}^T\mathbf{u}_h \right\|_{\mathbb{R}^M}^2,$$

with $\|\cdot\|_2$ the matrix 2-norm. Suppose that all the parameters affecting the full-order solver (e.g., the grid size) are fixed, so that the mass matrix $\mathbb{M}$ keeps unchanged as the size L of the reduced basis vary. Then, the error behaviour is entirely controlled by the term

$$\left\| \mathbf{u}_h - \mathbb{V}\mathbb{V}^T\mathbf{u}_h \right\|_{\mathbb{R}^M},$$

i.e., the distance (in the Euclidean norm) between $\mathbf{u}_h \in \mathbb{R}^M$ and its projection onto the column space $\mathrm{Col}\mathbb{V}$ of $\mathbb{V}$. In this respect, let us recall that, at the algebraic level, the reduced basis vectors $\{\boldsymbol{\psi}_i\}_{i=1}^L$ coincide with the first L left singular vectors of the snapshot matrix $\mathbb{S} = \left[\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \,\middle|\, \ldots \,\middle|\, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})\right] \in \mathbb{R}^{M\times N}$. From (2.61), we already know that for the columns of $\mathbb{S}$, i.e., the vectors collecting the degrees of freedom for the snapshots, the following holds:

$$\left\| \mathbf{u}_h(\boldsymbol{\mu}^{(n)}) - \mathbb{V}\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}^{(n)}) \right\|_{\mathbb{R}^M}^2 = \sum_{j=L+1}^{R} \sigma_j^2,$$

with R the rank of $\mathbb{S}$ and $\{\sigma_j\}_{j=1}^R$ its singular values. The above estimate can be generalized without any further hypothesis and upon small modifications to all the other elements in the snapshot manifold $\mathscr{M}_{\Xi_N}$. To be extendible to the entire discrete solution manifold $\mathscr{M}_h$, we have to assume that $\mathscr{M}_h$ is a low-dimensional subspace of V_h, so that the ensemble of snapshots is actually representative of the entire $\mathscr{M}_h$. Then, further assuming a rapid (i.e., exponential) decay of the singular values with the order, we have

$$\left\| u_h - u_h^{\mathbb{V}} \right\|_{L^2(\Omega)}^2 \approx \beta e^{-\alpha L}. \tag{2.76}$$

The last term involved in (2.74) regards the distance between $u_h^{\mathbb{V}}$ and $u_L^{\mathtt{NN}}$, i.e. the POD-NN solution. Similar to what is done for the previous term, recalling the definition (2.73) for $u_L^{\mathtt{NN}}$ we derive:

$$\begin{aligned}
\left\| u_h^{\mathbb{V}} - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} &= \int_\Omega \left| u_h^{\mathbb{V}} - u_L^{\mathtt{NN}} \right|^2 d\Omega = \int_\Omega \left| \sum_{i=1}^M \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h\right)_i \phi_i - \sum_{i=1}^M \left(\mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)_i \phi_i \right|^2 d\Omega \\
&= \sum_{i=1}^M \sum_{j=1}^M \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)_i \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)_j \underbrace{\int_\Omega \phi_i \phi_j \, d\Omega}_{\mathbb{M}_{i,j}} \\
&= \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)^T \mathbb{M} \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right) \\
&= \left\| \mathbb{M}^{1/2}\mathbb{V}\left(\mathbb{V}^T\mathbf{u}_h - \mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right) \right\|_{\mathbb{R}^M}^2 \\
&\leq \|\mathbb{M}\|_2 \|\mathbb{V}\|_2^2 \left\| \mathbb{V}^T\mathbf{u}_h - \mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} \right\|_{\mathbb{R}^L}^2 \leq L \|\mathbb{M}\|_2 \left\| \mathbb{V}^T\mathbf{u}_h - \mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} \right\|_{\mathbb{R}^L}^2,
\end{aligned} \tag{2.77}$$

where the last inequality follows from

$$\|\mathbb{V}\|_2 \leq \|\mathbb{V}\|_F = \sqrt{\mathrm{tr}\left(\mathbb{V}^T\mathbb{V}\right)} = \sqrt{\mathrm{tr}\left(\mathbb{I}_L\right)} = \sqrt{L}.$$

with $\|\cdot\|_F$ the Frobenius norm and $\mathbb{I}_L \in \mathbb{R}^{L\times L}$ the identity matrix of dimension L. In (2.77), by (2.72) the term

$$\left\| \mathbb{V}^T \mathbf{u}_h - \mathbf{u}_L^{\mathtt{NN}} \right\|_{\mathbb{R}^L} \tag{2.78}$$

coincides with the specific error function used to train the network. As for (2.75), this can be lowered to any given tolerance $\delta_{\mathtt{NN}}$, thanks to the Cybenko's result (see (ii) in Section 1.2.2), which ensures that one can always design (and train) a three-layer perceptron which approximates the map (2.71) to any desired level of accuracy, provided a sufficient number of training samples. However, as already pointed out in Section 1.2.4, no estimates on the convergence rate with respect to either the number of computing neurons or the dimension of the training set are available. Indeed, this is really problem dependent. Anyway, we shall see in the upcoming chapter that there exist some convenient situations in which the decay is surprisingly rapid. In this respect, let us notice that the factor $\sqrt{L}$ appearing in (2.78) suggests that the attaining of the desired accuracy gets harder as the number of POD modes, i.e., the dimension of the output space $\mathbb{R}^L$, increases.

Lastly, inserting (2.75), (2.76) and (2.77) into (2.74), we obtain the following estimate for the POD-NN error in the $L^2(\Omega)$-norm:

$$\left\| u - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} \leq \delta_{\mathtt{HF}} + \beta e^{-\alpha L} + \gamma \sqrt{L}\, \delta_{\mathtt{NN}}\,,$$

with α, β and γ positive constants, independent of both the solution and the size of the reduced basis, and $\delta_{\mathtt{HF}}$ and $\delta_{\mathtt{NN}}$ given tolerances. In other terms, the accuracy ensured by the POD-NN reduced basis method entirely relies on the accuracy of the underlying full-order solver, the quality of the reduced basis, and the accuracy of the approximation of the map (2.71).

Chapter 3

Numerical results

In this chapter, we present the numerical results obtained via the POD-G and the POD-NN RB methods applied to parametrized full-Dirichlet boundary value problems (BVPs) for the (nonlinear) Poisson equation and the incompressible steady Navier-Stokes equations. The problems which will be treated involve from one up to three parameters, either physical or geometrical. For the Poisson equation, we consider both linear and nonlinear problems stated on either one- or two-dimensional domains. In the one-dimensional case, we deal uniquely with physical parameters, which address the diffusion coefficient, the source term or the boundary conditions; whereas in two spatial dimensions we only consider parameters which affect the shape of the domain. For the two-dimensional Navier-Stokes equations we also consider a purely geometrical parametrization. Particularly, we will tackle the well-known lid-driven cavity problem for different values of the Reynold's number. The application of both the POD-G and the POD-NN techniques to differential problems featuring both physical and geometrical parameters, as well as natural or periodic boundary conditions, is left as future work.

The two RB techniques considered in this work are compared both in terms of accuracy and performance enabled at the online stage. For this, let $\mathbf{u}_L(\boldsymbol{\mu})$ and $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu})$ be respectively the (algebraic) POD-G and POD-NN solutions of reduced order L to the general $\boldsymbol{\mu}$-dependent problem (2.13). Furthermore, let $\|\cdot\|$ be the canonical (Euclidean) norm on $\mathbb{R}^M$. In the offline phases of the POD-G and POD-NN methods, for a new parameter value $\boldsymbol{\mu} \in \mathscr{P}$ (not involved either in the generation of the POD basis nor in the detection of an optimal neural network), the following *relative* errors with respect to the high-fidelity solution $\mathbf{u}_h(\boldsymbol{\mu})$ are analyzed:

(a) the POD-G relative error,

$$\varepsilon_{\mathtt{PODG}}(L,\boldsymbol{\mu}) = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_L(\boldsymbol{\mu})\right\|}{\left\|\mathbf{u}_h(\boldsymbol{\mu})\right\|} = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\,\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu})\right\|}{\left\|\mathbf{u}_h(\boldsymbol{\mu})\right\|}; \tag{3.1}$$

(b) the POD-NN relative error,

$$\varepsilon_{\mathtt{PODNN}}(L,\boldsymbol{\mu}) = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu})\right\|}{\left\|\mathbf{u}_h(\boldsymbol{\mu})\right\|} = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})\right\|}{\left\|\mathbf{u}_h(\boldsymbol{\mu})\right\|}; \tag{3.2}$$

(c) the relative projection error, i.e., the error committed by approximating the high-fidelity solution with its projection (in the discrete scalar product $(\cdot,\cdot)_h$, see (2.63))

onto the reduced space $V_{\mathtt{rb}}$,

$$\varepsilon_{\mathbb{V}}(L,\boldsymbol{\mu}) = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbf{u}_h^{\mathbb{V}}(\boldsymbol{\mu})\right\|}{\left\|\mathbf{u}_h(\boldsymbol{\mu})\right\|} = \frac{\left\|\mathbf{u}_h(\boldsymbol{\mu}) - \mathbb{V}\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\right\|}{\left\|\mathbf{u}_h(\boldsymbol{\mu})\right\|}; \tag{3.3}$$

recall that (3.3) provides a lower-bound for both (3.1) and (3.2) (as explained in Section 2.6.

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

$$\bar{\varepsilon}_{\mathtt{PODG}} = \bar{\varepsilon}_{\mathtt{PODG}}(L) = \frac{\sum_{\boldsymbol{\mu}\in\Xi_{te}} \varepsilon_{\mathtt{PODG}}(L,\boldsymbol{\mu})}{|\Xi_{te}|}, \qquad \bar{\varepsilon}_{\mathtt{PODNN}} = \bar{\varepsilon}_{\mathtt{PODNN}}(L) = \frac{\sum_{\boldsymbol{\mu}\in\Xi_{te}} \varepsilon_{\mathtt{PODNN}}(L,\boldsymbol{\mu})}{|\Xi_{te}|},$$

$$\bar{\varepsilon}_{\mathbb{V}} = \bar{\varepsilon}_{\mathbb{V}}(L) = \frac{\sum_{\boldsymbol{\mu}\in\Xi_{te}} \varepsilon_{\mathbb{V}}(L,\boldsymbol{\mu})}{|\Xi_{te}|},$$

with $|\Xi_{te}|$ the cardinality of Ξ_{te}.

Before diving into the results, let us give some details on the training phase for neural networks. While the training samples $\Xi_{tr} \subset \mathscr{P}$ are generated via successive latin hypercube samplings (as described in Algorithm 2.5), the validation inputs $\Xi_{va} \subset \mathscr{P}$ are randomly picked through a Monte Carlo sampling, as it is done for Ξ_{te}. The ratio between the size of Ξ_{va} and Ξ_{tr} is set to 0.3. The training is then performed using the multiple restarts approach to prevent the results to depend on the way the weights are (randomly) initialized (see Section 1.2.4). Typically, $K_{ms} = 5$ restarts are performed for each network topology. To ease the research for an optimal network configuration and by exploiting Cybenko's results (see (i) and (ii) in Section 1.2.2) we limit ourselves to two- and three-layers neural networks; in the latter case, we consider the same number of neurons for both hidden layers. For each topology, the hyperbolic tangent (see Fig. 1.4, on the right) is used as activation function for the hidden neurons.

Finally, let us discuss the choice of the number of POD modes L to be retained in the model. To this end, one could employ, e.g., the relative information content criterium (2.64). However, for the sake of analysis, we pursue a different approach, evaluating the errors (3.1), (3.2) and (3.3) (averaged over Ξ_{te}) for some (and often all) $L \le L_{max}$ (say $L_{max} = 40$). Observe that within the POD-NN framework, once a neural network has been (properly) trained for the approximation of the vectored-value map (2.71) with L components, then the approximation of the same map for any $\bar{L} < L$ comes freely by simply retaining the first $\bar{L}$ entries of the network output vector. Therefore, no additional training is required. Indeed, decorating the matrix $\mathbb{V}$ with a subscript denoting the dimension of the represented basis, one has

$$\mathbb{V}_{l+1} = \big[\underbrace{\boldsymbol{\psi}_1 \,\big|\, \dots \,\big|\, \boldsymbol{\psi}_l}_{\mathbb{V}_l} \,\big|\, \boldsymbol{\psi}_{l+1}\big] = \big[\mathbb{V}_l \,\big|\, \boldsymbol{\psi}_{l+1}\big] \quad \text{for any } l < M.$$

Hence:

$$\mathbb{V}_{l+1}^T \mathbf{u}_h(\boldsymbol{\mu}) = \begin{bmatrix} \mathbb{V}_l^T \\ \boldsymbol{\psi}_{l+1}^T \end{bmatrix} \mathbf{u}_h(\boldsymbol{\mu}) = \begin{bmatrix} \mathbb{V}_l^T \mathbf{u}_h(\boldsymbol{\mu}) \\ \boldsymbol{\psi}_{l+1}^T \mathbf{u}_h(\boldsymbol{\mu}) \end{bmatrix} \quad \text{for any } l < M.$$

Conversely, we have already discussed in Section 2.6 how the convergence provided by Newton's method may depend on L. Therefore, for a fair comparison with the POD-NN method we shall explicitly assemble and solve the reduced system for any $\boldsymbol{\mu} \in \Xi_{te}$ and any tested value of L.

3.1 One-dimensional Poisson equation

As a preliminar step before studying more complex problems in two dimensions, we consider BVPs for the one-dimensional Poisson equation. Three test cases are examined: the first linear and the others nonlinear, involving an increasing number of parameters.

3.1.1 Linear test case

We first consider the following linear Poisson equation in $\Omega = (-1,1)$:

$$-u(\mu)'' = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\mu}{2\sigma^2}\right), \tag{3.4}$$

with $\sigma = 0.2$ and $\mu \in [-1,1]$. The equation is completed with homogeneous Dirichlet boundary conditions. Let us first of all derive a POD-G RB model for (3.4) by generating a collection of linear finite elements discretizations $\{\mathbf{u}_h(\mu^{(1)}), \dots, \mathbf{u}_h(\mu^{(N)})\}$ over a (physical) uniform grid with 100 points. Here, it is not possible to use the latin hypercube sampling, since the parameter domain $\mathscr{P} \equiv [-1,1]$ is one-dimensional, hence we displace the training samples $\{\mu^{(1)}, \dots, \mu^{(N)}\}$ uniformly over $\mathscr{P}$. In the left plot of Fig. 3.1, we analyze the relative error between the FE and the POD-G method (averaged over a set Ξ_{te} of $N_{te} = 100$ parameter values) for various numbers N of snapshots. Independently of the number of snapshots used to generate the POD basis, we can appreciate an exponential decay of the error with

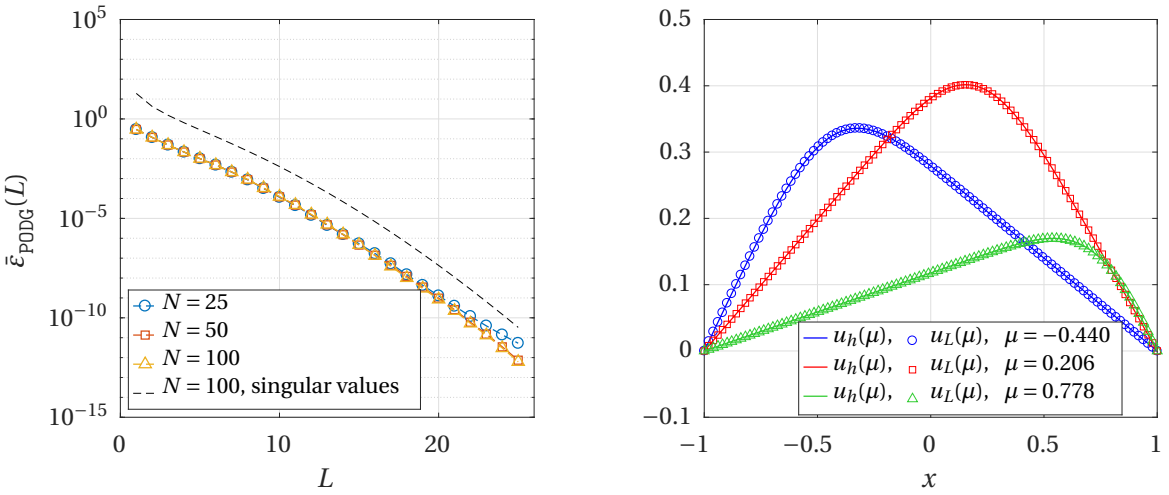


Figure 3.1. Left: error analysis for the POD-G RB method applied to (3.4); the results refer to a POD basis constructed based on $N = 25, 50, 100$ snapshots; for $N = 100$, the singular values of the corresponding snapshot matrix are also shown. Right: FE (solid line) and POD-G (markers) solutions for three different parameter values.

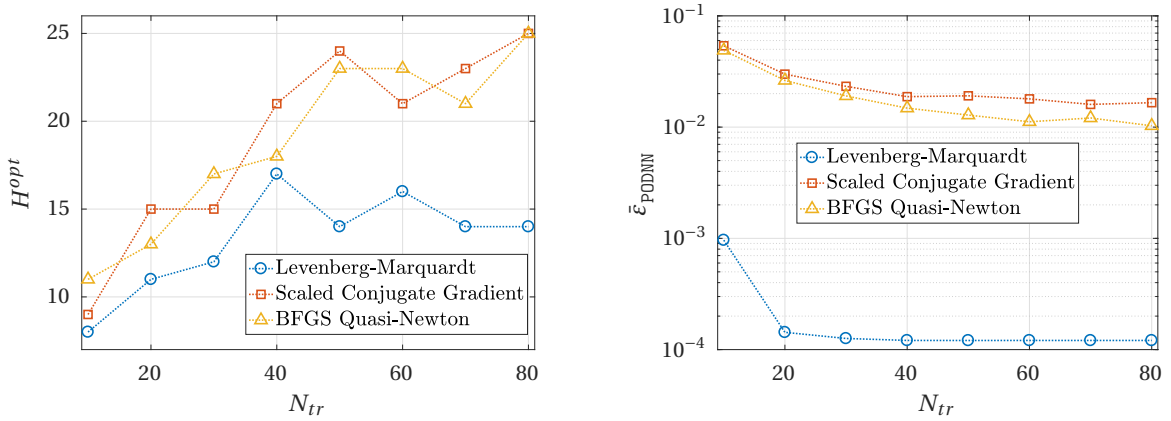


Figure 3.2. In respect to the approximation of the map (2.71) for the Poisson problem (3.4) via neural networks, comparison between three supervised algorithms - Levenberg-Marquardt (blue circles), Scaled Conjugate Gradient (orange squares), BFGS Quasi-Newton (yellow traingles) - in terms of the optimal configuration detected (*left*) and the associated (averaged) test error (*right*) for different amounts of training samples.

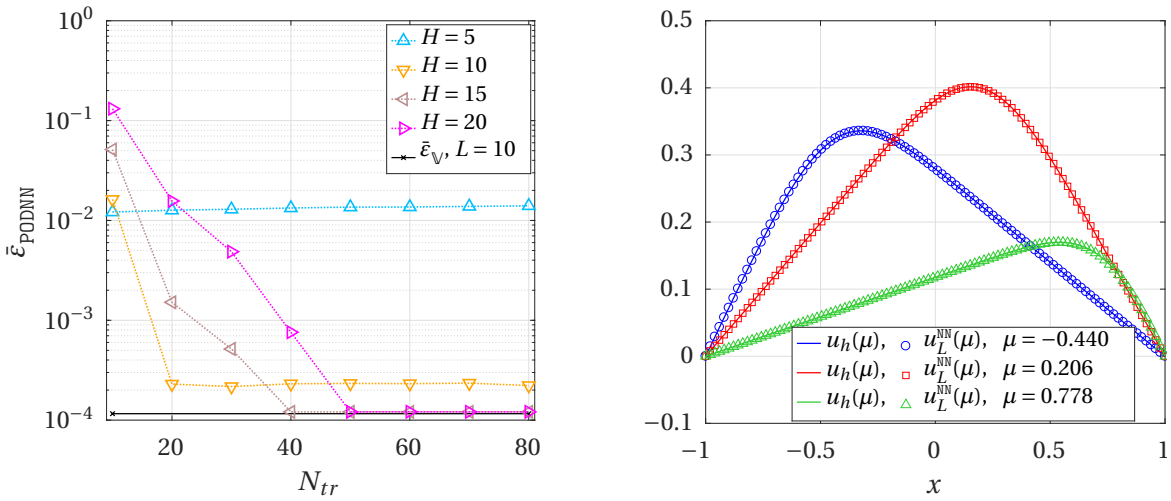


Figure 3.3. Left: relative error yielded by the POD-NN RB method applied to (3.4); the convergence to the projection error (solid line) for $L = 10$ is analyzed in terms of both the number of neurons included in the neural network and the dimension of the training set. Right: FE and POD-NN solutions for three parameter values.

the number L of POD modes retained by the reduced model. This behaviour is coherent with the one exhibited by the singular values. The effectiveness of the RB approximation is confirmed by the right plot of Fig. 3.1, which compares the FE and the POD-G solution for three values of μ, with $N = 50$ and $L = 10$.

Let us now move the analysis to the POD-NN method. To motivate our choice of using the Levenberg-Marquardt learning algorithm (see Section 1.2.3), we compare it with two other well-known and widely-used supervised training routines: the Scaled Conjugate Gradient [45] and the BFGS Quasi-Newton method [21]. For this, we consider the approxi-

mation of the map (2.71) with $L = 10$ POD modes, i.e., $\mathbb{V} \in \mathbb{R}^{M\times 10}$. For this simple linear case, we can limit ourselves to networks with a single hidden layer, embodying H inner neurons, with $H \in \{5, 10, 15, 20, 25\}$. For a given number N_{tr} of training samples, Fig. 3.2 tracks the number of hidden processing units required to yield the minimum error on Ξ_{te} (*left*) and the optimal error itself (*right*) for each one of the three algorithms. We can observe that, for any tested value of N_{tr}, the Levenberg-Marquardt algorithm (blue circles) leads to much more accurate approximations using fewer neurons. In other words, it turns out to be not only effective but even efficient. A further confirmation is given the decay of the POD-NN error as the number of employed training patterns N_{tr} and neurons H increases, as shown in the left plot of Fig. 3.3. Here, the lower-bound given by the projection error is attained via a network with $H = 15$ hidden neurons, exposed to $N_{tr} = 40$ training samples. The solutions obtained by such network are provided in the right plot of Fig. 3.3 and they substantially overlap with the corresponding FE solutions.

3.1.2 Nonlinear test case, two parameters

The first nonlinear problem we consider is:

$$\begin{cases} -\left(\left(1+u(\boldsymbol{\mu})^2\right)\, u(\boldsymbol{\mu})'\right)' = s(x;\boldsymbol{\mu}) \quad \text{in } \Omega = [-\pi,\pi]\,, \\ u(\boldsymbol{\mu})\big|_{x=-\pi} = \mu_2 \sin(2-\mu_1\,\pi)\,, \quad u(\boldsymbol{\mu})\big|_{x=\pi} = \mu_2 \sin(2+\mu_1\,\pi)\,, \end{cases} \tag{3.5}$$

where $\boldsymbol{\mu} = \left(\mu_1, \mu_2\right) \in \mathscr{P} = [1,3]\times[1,3]$ and the source term $s(x;\boldsymbol{\mu})$ is chosen so that $u_{ex}(x;\boldsymbol{\mu}) = \mu_2 \sin(2+\mu_1\, x)$ is the exact solution to the problem for any $\boldsymbol{\mu} \in \mathscr{P}$. Here, a POD-Galerkin RB model built upon $N = 50$ samples (randomly picked via LHS) is highly effective. Indeed, as reported in Fig. 3.4, the relative error averaged over $N_{te} = 100$ samples rapidly decades as the number of retained POD basis functions L increases, attaining a value of 10^{-7} already for $L = 15$. Hence, the design of an effective neural network, which can at least approach the accuracy provided by the algebraic ROM, is really challenging. The results concerning the search for an optimal three-layers network configuration using the methodology described in Algorithm 2.5 are provided in the left plot of Fig. 3.5. The steps followed by the routine

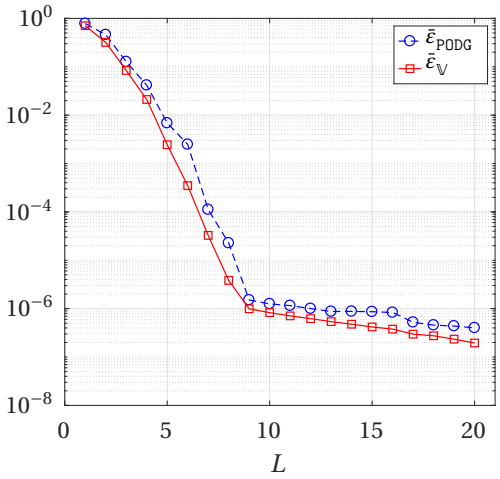


Figure 3.4. Error analysis for the POD-Galerkin RB method applied to the problem (3.5). The lower-bound provided by the projection error (3.3) is reported as reference.

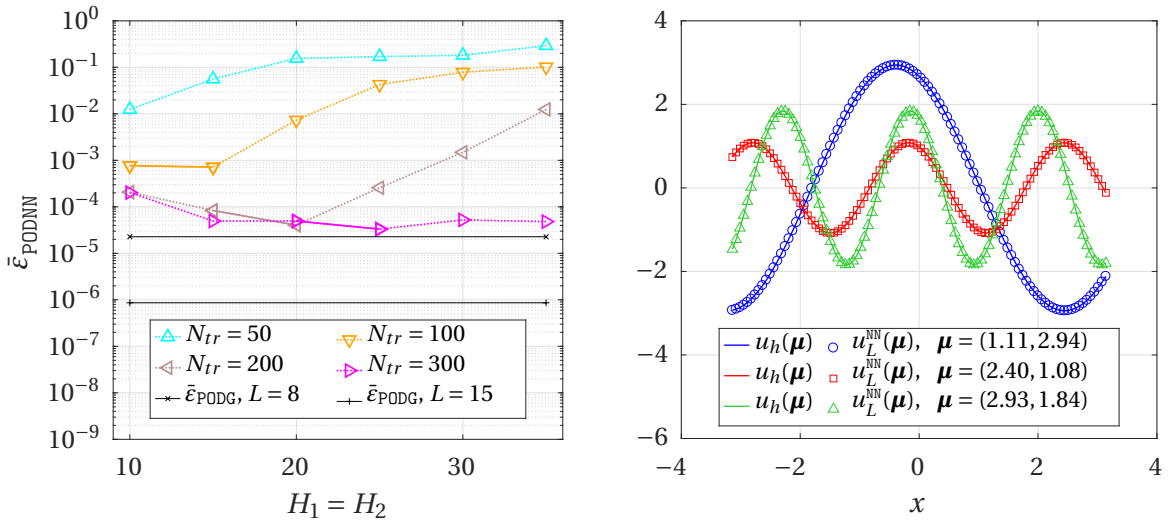


Figure 3.5. Left: error analysis for the POD-NN RB method (dotted lines) applied to problem (3.5) for several numbers of training samples. The solid sections represent the stepsfollowed by the Algortithm 2.5; the error yielded by the POD-G method using $L = 8$ or $L = 15$ basis functions are reported as reference. Right: coherence of FE and POD-NN solutions for three input vectors $\boldsymbol{\mu}$.

are represented by solid tracts. However, for the sake of completeness, we also report the test error (i.e., the minimum over multiple restarts) for any considered number of hidden neurons per layer (H_1 and H_2) and any dimension of the training set (N_{tr}).

Let us first point out that the basic assumption which Algorithm 2.5 relies on is fulfilled: as the number of available samples increases, the amount of neurons which allows to attain the minimum error increases as well, while the minimum error itself decreases. Although the networks have been trained to approximate $L = 15$ projection coefficients, we can not guarantee the same precision obtained by the POD-G method retaining the same amount of basis functions. Indeed, using POD-NN with $L = 15$ basis vectors we retrieve the same accuracy of POD-G employing only $L = 8$ basis vectors. To get some insights on this, consider the regression plots in Fig. 3.6, where we compare, for an MLP with $H_1 + H_2 = 50$ hidden neurons and for each test sample in Ξ_{te}, some of the entries of the network output with the corresponding target expansion coefficients. While the first and seventh coefficients (*top*) are interpolated almost exactly, bad fitting problems arise for the eighth (*bottom left*) and especially for the nineth (*bottom right*) ones. This suggests that the Levenberg-Marquardt algorithm is sufficiently *smart* to detect that the average magnitude of a projection coefficient decreases with the ordinality of the coefficient itself [1], and so that particular effort should be put in properly approximating the first expansion coefficients. Indeed, although the prediction for the nineth entry of the vector $\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})$ is completely wrong in most of the cases, the average value of this component is around 10^{-4}. As a result, the global solution provided by the network is really close to the corresponding FE discretization, as shown in Fig. 3.5 (*right*).

[1] This observation follows from the fact that the energy retained by each POD mode $\boldsymbol{\psi}_i$ is monotonically decreasing in i.

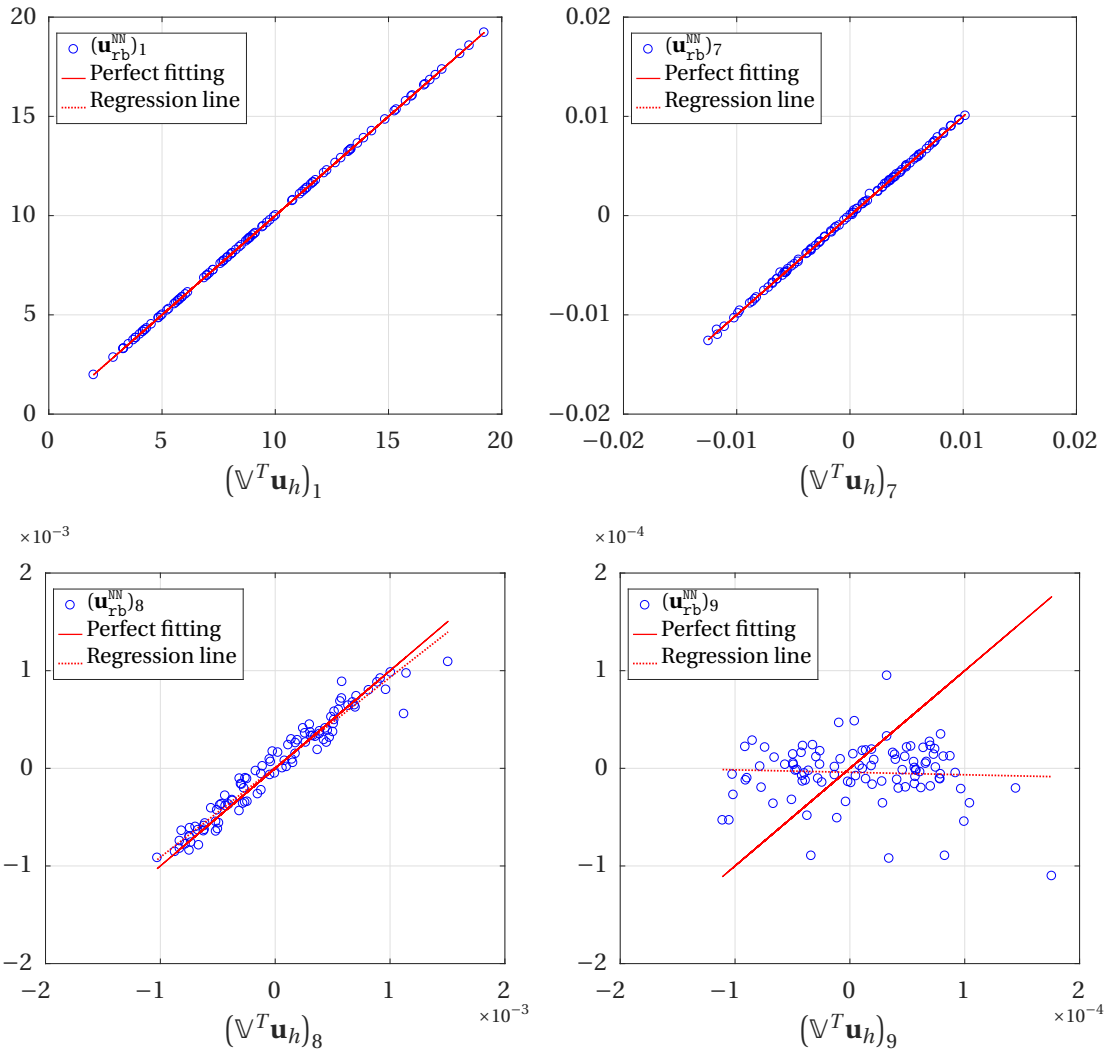


Figure 3.6. From left to right, top to bottom: regression plots for the first, seventh, eighth and nine scalar components of the outputs provided on Ξ_{te} by a network with $H_1 = H_2 = 25$ neurons per hidden layer, trained to approximate the map (2.71) for problem (3.5).

3.1.3 Nonlinear test case, three parameters

The last problem we consider for the one-dimensional Poisson equation features an exponential nonlinearity (in the diffusion coefficient) and involves three parameters:

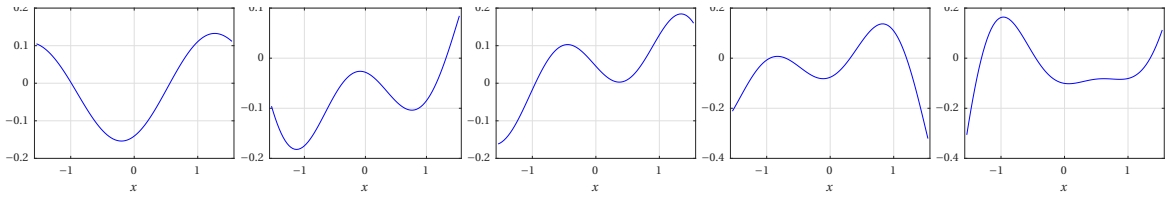
$$\begin{cases} -\big(\exp(u(\boldsymbol{\mu}))\ u(\boldsymbol{\mu})'\big)' = s(x;\boldsymbol{\mu}) \quad \text{in } \Omega = \left(-\dfrac{\pi}{2}, \dfrac{\pi}{2}\right), \\ u(\boldsymbol{\mu})\big|_{x=\pm\pi/2} = \mu_2\left(2 \pm \sin\left(\dfrac{\mu_1\pi}{2}\right)\right)\exp\left(\pm\dfrac{\mu_3\pi}{2}\right), \end{cases} \tag{3.6}$$

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

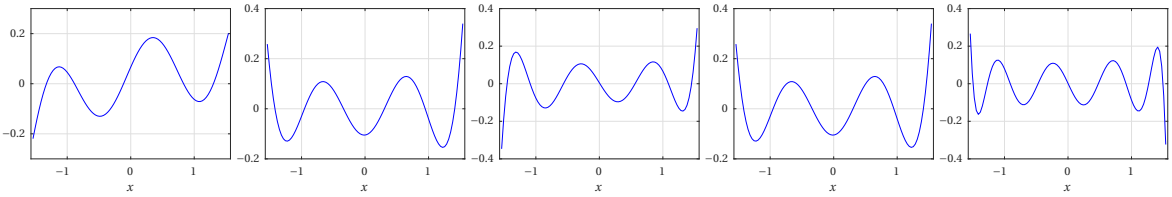
$$u_{ex}(x;\boldsymbol{\mu}) = \mu_2\big(2 + \sin(\mu_1 x)\big)\exp\big(\mu_3 x\big)$$

is the exact solution to the problem for any $\boldsymbol{\mu} \in \mathscr{P}$. Figure 3.7 represents the first 10 POD modes for the problem, yielded by singular value decomposition of $N = 100$ (linear) FE

snapshots computed over a uniform grid of 100 nodes. Their inclusion within the standard POD-Galerkin RB framework enables an average relative error of about 10^{-5} on a test set Ξ_{te} of $N_{te} = 100$ randomly chosen samples. This is whitnessed by Fig. 3.8, which shows the convergence of the solution with respect to the number of basis functions L retained in the model (*left*). The results for the POD-NN method employing three-layers neural networks with a varying number of neurons H_i, $i = 1, 2$, and learning patterns N_{tr}, are also provided, together with the comparison with the FE solutions for some parameter values (*right*). Note that, when only a few modes are considered (e.g., $L \leq 5$), the POD-NN scheme may lead to a significant reduction of the error with respect to the POD-G technique. Moreover, as we expand the training set, we obtain more accurate predictions for a larger number of POD coefficients, provided that we allow the size of the network to increase.



(a) ψ_1 **(b)** ψ_2 **(c)** ψ_3 **(d)** ψ_4 **(e)** ψ_5



(f) ψ_6 **(g)** ψ_7 **(h)** ψ_8 **(i)** ψ_9 **(j)** ψ_{10}

Figure 3.7. The first 10 POD basis functions for the (nonlinear) Poisson problem (3.6).

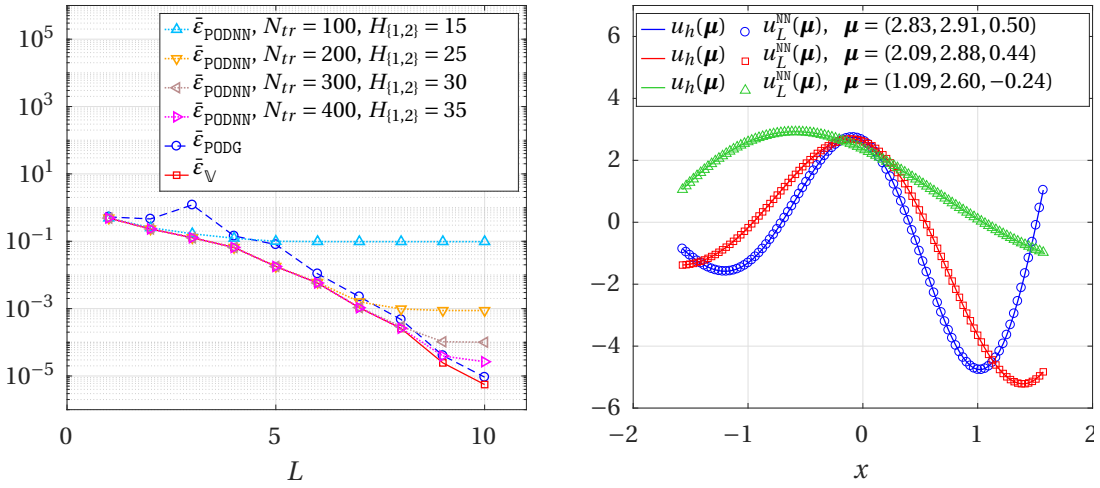


Figure 3.8. Convergence analysis for the POD-G and POD-NN methods applied to problem (3.6) (*left*) and comparison between the FE and the POD-NN solutions for three parameter values (*right*). These latter results have been obtained via a neural network with $H_1 = H_2 = 35$ units per hidden layer, and considering $L = 10$ POD modes.

Observe that the results shown in the left plot of Fig. 3.8 refer to neural networks trained with the set of coefficients

$$\{(u_h(\boldsymbol{\mu}), \psi_i)_h\}_{i=1}^{\bar{L}} \equiv \{(\mathbf{u}_h(\boldsymbol{\mu}), \boldsymbol{\psi}_i)_{\mathbb{R}^M}\}_{i=1}^{\bar{L}}$$

for each learning sample $\boldsymbol{\mu} \in \Xi_{tr}$, with $\bar{L} = 10$. Therefore, each network is equipped with $\bar{L} = 10$ output neurons - one for each coefficient. Once the training has been successfully completed, given a new input $\boldsymbol{\mu} \in \mathscr{P}$, the corresponding POD-NN RB solution of order $\bar{L}$ is obtained as:

$$\mathbf{u}_{\bar{L}}^{\mathtt{NN}}(\boldsymbol{\mu}) = \sum_{i=1}^{\bar{L}} \left(\mathbf{u}_{\bar{L}}^{\mathtt{NN}}\right)_i \boldsymbol{\psi}_i \,, \tag{3.7}$$

where $\left(\mathbf{u}_{\bar{L}}^{\mathtt{NN}}\right)_i$ denotes the network response observed at the i-th output neuron, $i = 1, \ldots, \bar{L}$. Although $\mathbf{u}_{\bar{L}}^{\mathtt{NN}}(\boldsymbol{\mu})$ represents the best prediction for $\mathbf{u}_h(\boldsymbol{\mu})$ provided by the network, we can also construct lower-ordered approximations $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu})$, with $L < \bar{L}$, by simply retaining the first L terms in the summation. The claim is that such L-order RB solution is basically as accurate as the one we could obtain by exposing the network to exactly L POD coefficients per training sample. In other terms, employing a larger amount of modes during the training and a smaller one in the subsequent online queries does not enable a substantial improvement in the performace. A confirmation of this is given in Fig. 3.9, where the average relative POD-NN error $\bar{\varepsilon}_{\mathtt{PODNN}}(L)$, for $1 \leq L \leq 10$, is shown for a network equipped with either 20 or 35 neurons per hidden layer, and exposed to $N_{tr} = 300$ samples and either L or $\bar{L}$ expansion coefficients per sample.

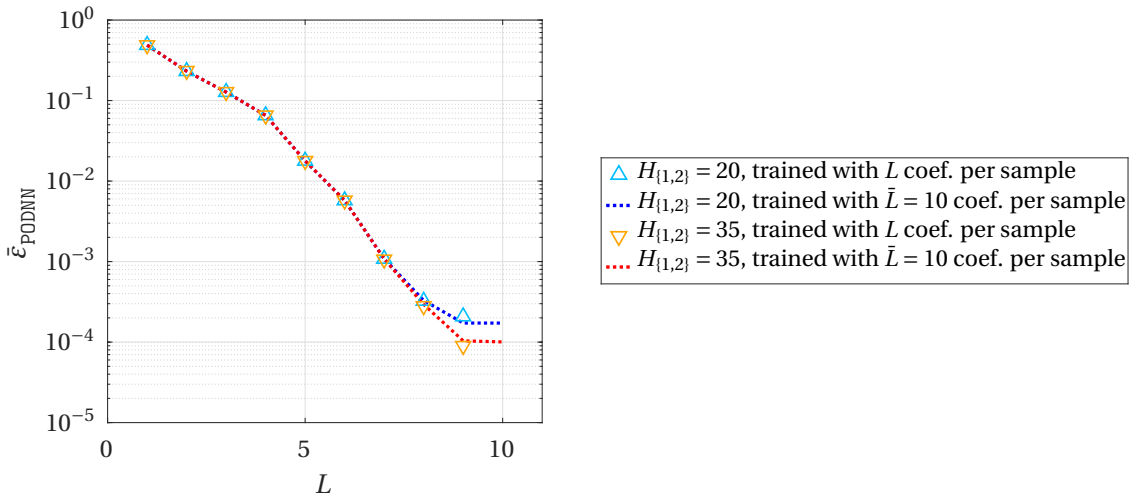
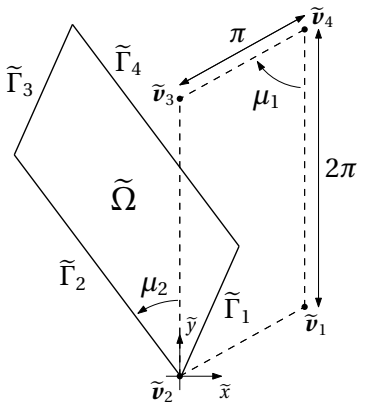


Figure 3.9. Sensitivity analysis for the POD-NN method applied to problem (3.6) with respect to the number of POD coefficients (per sample) used during the training; to avoid overfitting, $N_{tr} = 300$ learning patterns have been used.

3.2 Two-dimensional Poisson equation

As mentioned in the introductory part, the two-dimensional differential problems we deal with uniquely involve a geometrical parametrization, that is, $\boldsymbol{\mu} \equiv \boldsymbol{\mu}_g$. In particular, in this section we address the numerical discretization via RB methods of three BVPs concerning the linear and nonlinear Poisson equation. Each problem will be stated over one of the two domains $\widetilde{\Omega}(\boldsymbol{\mu})$ given in Fig. 3.10 and 3.11. The former consists of a parallelogram, possibly rotated around the origin, characterized by two parameters: one of the internal angles, $\mu_1 \in \left[\pi/4, 3\pi/4\right]$, and the rotation angle itself, $\mu_2 \in \left[0, \pi/2\right]$. The latter represents a stenosis geometry, parametrized in the depths μ_1 and μ_2 of the bottom and top restrictions (or inflations), respectively, and the length μ_3 of the vessel; the parameter domain is given by $\mathscr{P} = [-0.5, 0.5] \times [-0.5, 0.5] \times [1, 5]$. For both geometries, the boundary parametrizations are reported alongside their visualization.

As discussed in Section 2.2, in order to generate snapshots which can then be combined and processed via POD, the problems have to be re-formulated and solved on a reference, i.e., parameter-independent, domain Ω, discretized through a triangular mesh Ω_h with M (inner) vertices. Here, Ω corresponds to the unit square shown in Fig. 2.2 and the volumetric transformation $\boldsymbol{\Phi} : \Omega \times \mathscr{P} \to \widetilde{\Omega}$, used to transpose the PDE from the physical to the fixed domain, is provided by the boundary displacement-dependent transfinite map (2.30) (see Section 2.2.3).



$$\mathbb{Q}(\mu_2) = \begin{bmatrix} \cos(\mu_2) & -\sin(\mu_2) \\ \sin(\mu_2) & \cos(\mu_2) \end{bmatrix}$$

$$\widetilde{\boldsymbol{v}}_1 = \widetilde{\boldsymbol{v}}_1(\mu_1) = \left[\pi \sin(\mu_1), \pi \cos(\mu_1)\right]^T$$

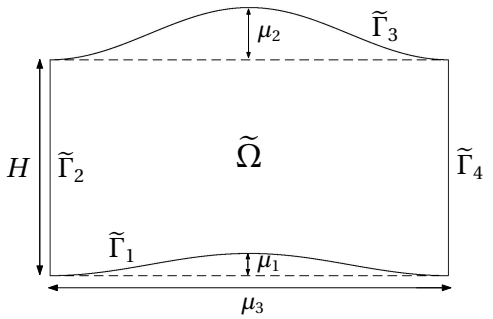
$$\widetilde{\boldsymbol{v}}_2 = \widetilde{\boldsymbol{v}}_2(\mu_1) \equiv \left[0, 0\right]^T$$

$$\widetilde{\boldsymbol{v}}_3 = \widetilde{\boldsymbol{v}}_3(\mu_1) \equiv \left[0, 2\pi\right]^T$$

$$\widetilde{\boldsymbol{v}}_4 = \widetilde{\boldsymbol{v}}_4(\mu_1) = \left[\pi \sin(\mu_1), \pi\big(2 + \cos(\mu_1)\big)\right]^T$$

$$\widetilde{\boldsymbol{\Psi}}_i(t; \boldsymbol{\mu}) = \mathbb{Q}(\mu_2)\left[t\,\widetilde{\boldsymbol{v}}_i + (1-t)\,\widetilde{\boldsymbol{v}}_{i+1}\right] \quad \text{for } 1 \le i \le 4, \text{ with } \widetilde{\boldsymbol{v}}_5 \equiv \widetilde{\boldsymbol{v}}_1$$

Figure 3.10. The quadrilateral domain used in the simulations (*left*, solid line) and the parametrizations of its sides (*right*).



$$\widetilde{\boldsymbol{\Psi}}_1(t; \boldsymbol{\mu}) = \left[\mu_3\,(1-t),\, \mu_1 \cos\left(2\pi t - \frac{\pi}{2}\right) + \mu_1\right]^T$$

$$\widetilde{\boldsymbol{\Psi}}_2(t; \boldsymbol{\mu}) \equiv \left[0,\, H\,t\right]^T$$

$$\widetilde{\boldsymbol{\Psi}}_3(t; \boldsymbol{\mu}) = \left[\mu_3\,t,\, H - \mu_2 \cos\left(2\pi t - \frac{\pi}{2}\right) - \mu_2\right]^T$$

$$\widetilde{\boldsymbol{\Psi}}_4(t; \boldsymbol{\mu}) = \left[0,\, H(1-t)\right]^T$$

Figure 3.11. The stenosis geometry employed in the simulations (*left*) and the parametrizations of its sides (*right*).

3.2.1 Linear test case

Let $\widetilde{\Omega} = \widetilde{\Omega}(\boldsymbol{\mu})$ be the stenosis geometry shown in Fig. 3.11, with $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in \mathscr{P} = [-0.5, 0.5] \times [-0.5, 0.5] \times [1, 5]$ and consider the following linear Poisson problem for $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$:

$$\begin{cases} -\widetilde{\Delta}\widetilde{u}(\boldsymbol{\mu}) = \widetilde{s}(\widetilde{x}, \widetilde{y}) = 2\sin(\widetilde{x})\cos(\widetilde{y}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \\ \widetilde{u}(\boldsymbol{\mu}) = \sin(\widetilde{\sigma}_x)\cos(\widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu}). \end{cases} \tag{3.8}$$

After restating the problem over the reference domain Ω by means of the map (2.31) and the change of variables formulae reported in Section 2.2.1, the FE discretization yields the linear system

$$\mathbb{A}(\boldsymbol{\mu})\,\mathbf{u}_h(\boldsymbol{\mu}) = \mathbf{s}(\boldsymbol{\mu}) \tag{3.9}$$

in the nodal evaluations $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$ of the high-fidelity solution $u_h(\boldsymbol{\mu})$. The matrix $\mathbb{A}(\boldsymbol{\mu}) \in \mathbb{R}^{M\times M}$ and the vector $\mathbf{s}(\boldsymbol{\mu}) \in \mathbb{R}^M$ are defined as

$$\mathbb{A}(\boldsymbol{\mu})_{i,j} = \sum_{K\in\Omega_h} a_K(\phi_j, \phi_i) = \int_K \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_j \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i \left|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\right| dK \quad \text{for } i,\, j = 1, \dots, M,$$

$$\mathbf{s}(\boldsymbol{\mu})_i = \sum_{K\in\Omega_h} \int_K \left(\widetilde{s}\circ\boldsymbol{\Phi}\right)\phi_i \left|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\right| dK - \sum_{K\in\Omega_h} a_K(l_h(\boldsymbol{\mu}), \phi_i) \qquad \text{for } i = 1, \dots, M.$$

Here, we recall that K denotes a generic triangular element of the computational mesh Ω_h, $\{\phi_i\}_{i=1}^M$ is the canonical (Lagrangian) FE basis for $V_h = X_h^1 \cap H_0^1(\Omega)$, $\mathbb{J}_{\boldsymbol{\Phi}} : \Omega \times \mathscr{P} \to \mathbb{R}^{2\times 2}$ represents the Jacobian (with respect to the spatial coordinates) of $\boldsymbol{\Phi}(\cdot;\boldsymbol{\mu})$, and $l_h(\boldsymbol{\mu}) \in X_h^1$ is the (discrete) lifting function. Therefore, letting $\mathbb{V} \in \mathbb{R}^{M\times L}$ represent a POD basis of rank L, for a given $\boldsymbol{\mu} \in \mathscr{P}$ the Galerkin projection procedure leads to the reduced system

$$\mathbb{A}_L(\boldsymbol{\mu})\,\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) = \mathbf{s}_L(\boldsymbol{\mu})$$

in the expansion coefficients $\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^L$, where

$$\mathbb{A}_L(\boldsymbol{\mu}) = \mathbb{V}^T\mathbb{A}(\boldsymbol{\mu})\mathbb{V} \in \mathbb{R}^{L\times L} \quad \text{and} \quad \mathbf{s}_L(\boldsymbol{\mu}) = \mathbb{V}^T\mathbf{s}(\boldsymbol{\mu}) \in \mathbb{R}^L.$$

Hence, the assembly of the reduced-order matrices $\mathbb{A}_L(\boldsymbol{\mu})$ and $\mathbf{s}_L(\boldsymbol{\mu})$ requires their full-order counterpart $\mathbb{A}(\boldsymbol{\mu})$ and $\mathbf{s}(\boldsymbol{\mu})$. However, the (expensive) construction of these latter terms can be performed exclusively at the online stage, due to the non-affine dependence of the map $\boldsymbol{\Phi}(\cdot;\boldsymbol{\mu})$ on $\boldsymbol{\mu}$, induced by the parametrizations $\boldsymbol{\Psi}_i(\cdot;\boldsymbol{\mu})$, $i = 1, 2, 3, 4$, characterizing the boundaries of the original domain $\widetilde{\Omega}(\boldsymbol{\mu})$ (see Fig. 3.11). This clearly limits the advantages of the POD-Galerkin scheme of solving a reduced system. In this respect, let us consider Fig. 3.12, reporting the online run times for the FE, POD-G and POD-NN methods applied to (3.8) for $N_{te} = 100$ randomly picked values of $\boldsymbol{\mu}$. The times refer to a MATLAB® implementation of the three methods in exam, and have been measured on a laptop equipped with an Intel Core i7-6498DU @ 2.50 GHz processor. Both the RB models include $L = 15$ basis functions, constructed via proper orthogonal decomposition of $N = 100$ snapshots, computed over a triangular grid with $M = 16961$ internal vertices. Although the Galerkin projection over the reduced space leads to a reduction in the system size by a factor of 10^3, no computational saving is attained for any test sample. Conversely, the POD-NN method enables an average online speed up of about 400. This comes at the cost of an extended offline phase. Indeed, the automatic routine 2.5, performed to find an optimal combination

of the number of training samples and hidden neurons to use, takes around 2 hours, as opposed to the few minutes required to build $N = 100$ FE snapshots (through the resolution of the system (3.9)) and compute their singular value decomposition. However, coupling POD with a neural network with $H_1 = H_2 = 35$ computing units per hidden layer and whose weights are adjusted relying upon $N_{tr} = 200$ training patterns, yields (actually slightly) more accurate results than the POD-G method does, as can be inferred from Fig. 3.13 (*left*). In particular, the POD-NN error $\bar{\varepsilon}_{\texttt{PODNN}}(L)$ matches the projection error $\bar{\varepsilon}_{\mathbb{V}}(L)$ for any $L \leq 15$. A convergence analysis for the POD-NN method with respect to the number of hidden neurons H_i, $i = 1, 2$, and the size N_{tr} of the training data set Ξ_{tr} is given in the right panel of Fig. 3.13.

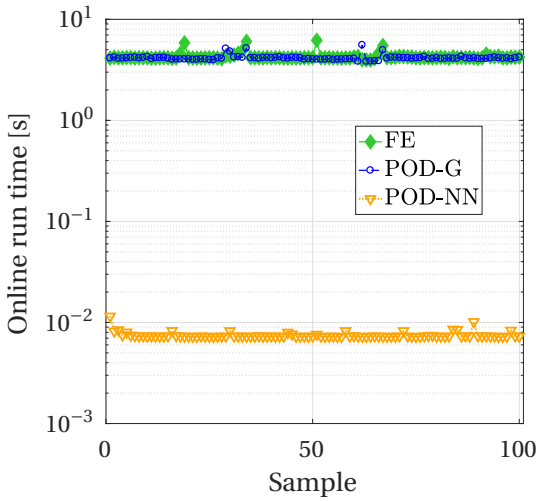


Figure 3.12. Online run times for the FE, POD-G and POD-NN methods applied to problem (3.8), for $N_{te} = 100$ randomly generated parameter values.

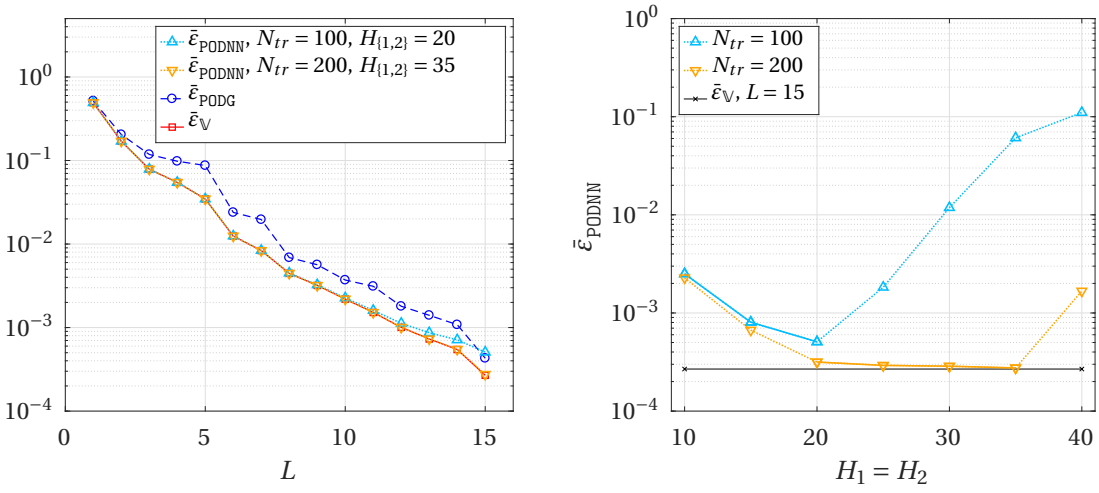


Figure 3.13. On the left, error analysis for the POD-G (blue dashed line) and the POD-NN (dotted lines) methods applied to problem (3.8). For the latter, a convergence analysis with respect to the number of training samples and hidden neurons used is provided on the right. The solid tracts denote the steps followed by the automatic training routine 2.5. All the results refer to neural networks exposed to $L = 15$ generalized coordinates per learning sample.

3.2.2 Nonlinear test case, two parameters

Let us address the upcoming BVP for a semilinear Poisson equation, stated over the parametrized quadrilateral domain $\widetilde{\Omega}(\boldsymbol{\mu})$ illustrated in Fig. 3.10:

$$\begin{cases} -\widetilde{\nabla}\cdot\left((1+\widetilde{u}(\boldsymbol{\mu})^2)\,\widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu})\right) = \widetilde{s}(\widetilde{x},\widetilde{y}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{\sigma}_y \sin(\widetilde{\sigma}_x)\cos(\widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu}). \end{cases} \tag{3.10}$$

Here, the source term $\widetilde{s} = \widetilde{s}(\widetilde{x},\widetilde{y})$ has been chosen so that for any $\boldsymbol{\mu} \in \mathscr{P} = \left[\pi/4, 3\pi/4\right] \times \left[0, \pi/2\right]$ the exact solution to the problem is given by $\widetilde{u}_{ex}(\widetilde{x},\widetilde{y}) = \widetilde{y}\sin(\widetilde{x})\cos(\widetilde{y})$. We have already referred to this problem in Section 2.6, as it lays bare the major weak point of the standard POD-Galerkin framework. In fact, as illustrated by Fig. 2.4, although accurate, the POD-G method only seldomly leads to a computational saving at the online stage with respect to the FE scheme, due to the nonaffine parametric dependence of the underlying BVP. Instead, the proposed POD-NN RB procedure enables a substantial speed-up while guaranteeing high predictive precision. At this regard, Fig. 3.14 compares the truth (*left*), POD-G (*center*) and POD-NN (*right*) solutions to problem (3.10) for $\boldsymbol{\mu} = (0.835, 0.034)$. The FE discretization has been built over a mesh consisting of $M = 1921$ nodes, whereas the RB solutions have been constructed combining $L = 35$ POD basis functions, given by singular value decomposition of $N = 100$ snapshots. In particular, the POD-NN approximation has been obtained through a three-layers perceptron endowed with $H_1 + H_2 = 70$ hidden neurons and trained with $N_{tr} = 300$ learning patterns. Both RB frameworks turn out to be powerful predictive tools, as whitnessed by the good agreement between the corresponding solutions and the full-order one. This is further confirmed by the left plot in Fig. 3.15, which reports the POD-G error (3.1) and the POD-NN error (3.2) for $N_{te} = 50$ randomly picked samples not involved either in the generation of the reduced basis, nor in the networks training. However, using neural networks not only yields precise responses, but also drastically reduces the time required to tackle any online query. Indeed, the right panel in Fig. 3.15, which shows the online run times on Ξ_{te} for the two RB methodologies in exam, reveals how the POD-NN method is about 10^3 times faster than POD-G. Yet, the search for the optimal network configuration through the routine outlined in Algorithm 2.5 unavoidably extends the offline stage of the POD-NN method. Here, it lasts around 2.5 hours, of which only 18 minutes are devoted to the generation of the reduced basis. Hence, considering that the FE scheme applied to (3.10) takes (on average) 10 seconds per query, the POD-NN procedure should be prefered to a direct approach only when the problem has to be solved for more than 900 different input vector parameters.

Lastly, consider Fig. 3.16, which provides a convergence analysis for the POD-NN RB scheme with respect to the number N_{tr} of samples used to train the network (*left*) and the number $H_1 = H_2$ of internal neurons the network is endowed with (*right*). For the latter plot, the results have been obtained by means of $N_{tr} = 300$ training patterns - not to incur in overfitting. Let us remark that, as for the POD-Galerkin procedure, for any fixed N_{tr} and H_i, $i = 1, 2$, the error yielded by the proposed method is monotonically decreasing in the number L of retained basis functions, approaching or even attaining the projection error for each tested value of L, provided an oculate choice for N_{tr} and H_i, $i = 1, 2$.

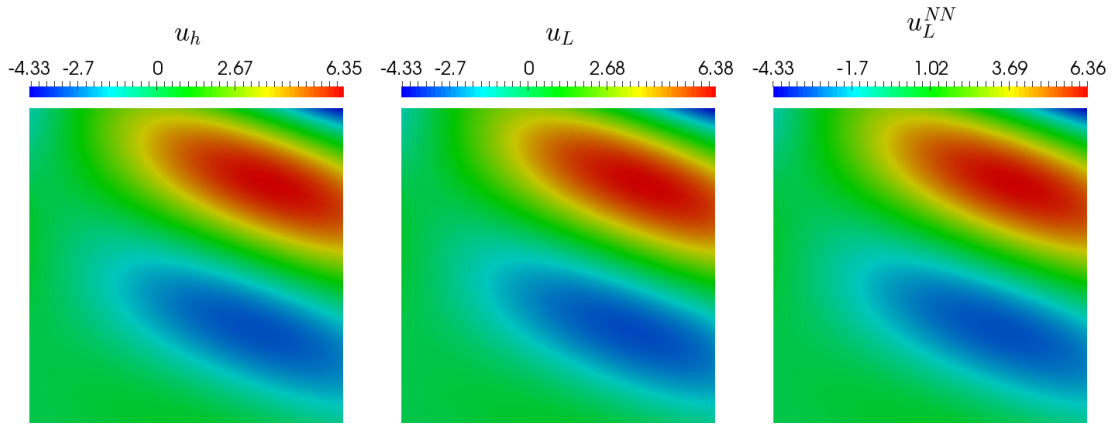


Figure 3.14. Finite element (*left*), POD-G (*center*) and POD-NN (*right*) solution to the semilinear Poisson problem (3.10) re-stated over the reference domain Ω, with $\boldsymbol{\mu} = (0.835, 0.034)$.

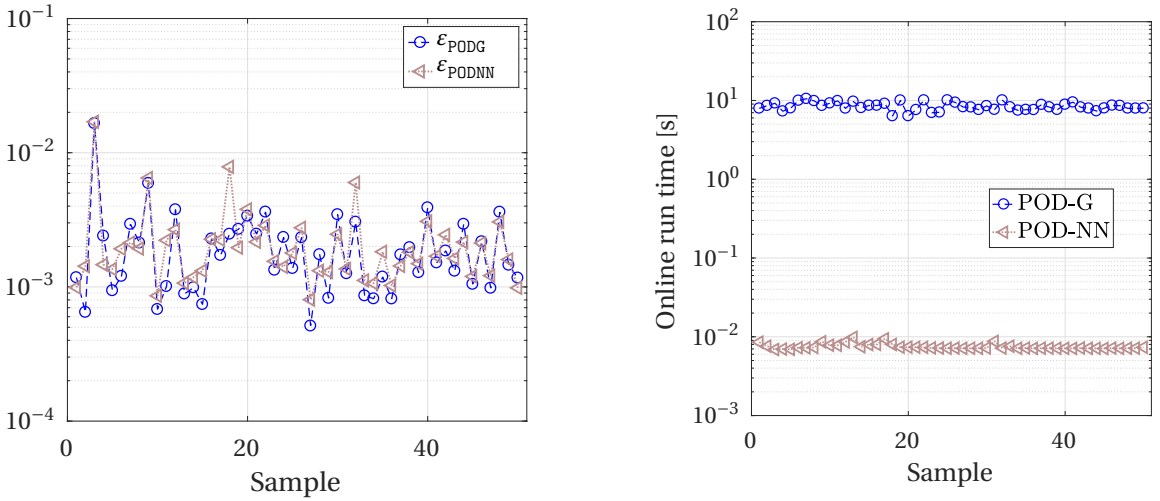


Figure 3.15. Relative errors (*left*) and online run times (*right*) for the POD-G and the POD-NN methods applied to problem (3.10) for $N_{te} = 50$ randomly picked parameter values.

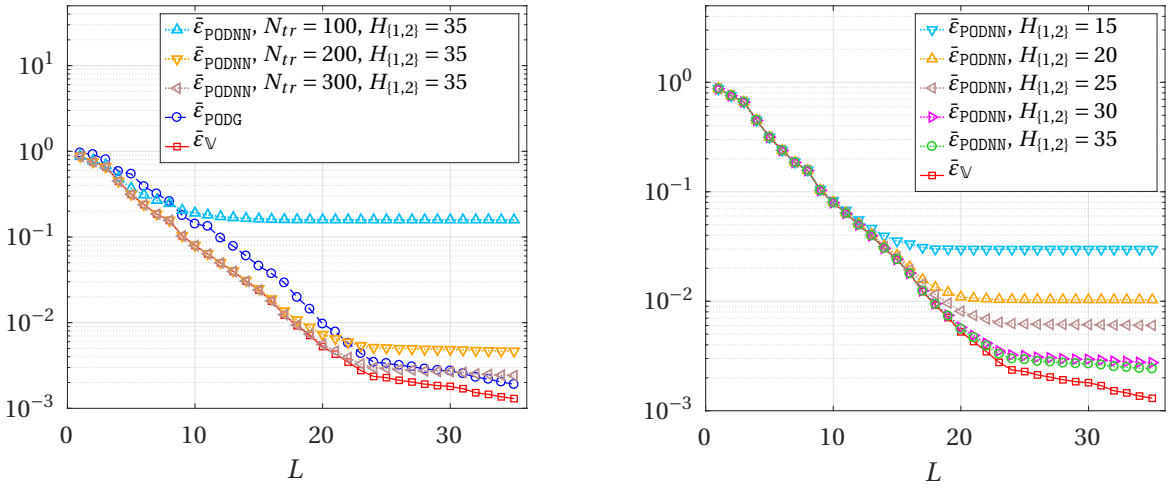


Figure 3.16. Convergence analysis for the POD-NN RB method applied to the BVP (3.10). The results have been obtained via three-layers perceptrons with $H_1 = H_2 \in \{15, 20, 25, 30, 35\}$ neurons per hidden layer and trained with $N_{tr} \in \{100, 200, 300\}$ learning patterns. Each pattern consists of an input vector $\boldsymbol{\mu} \in \mathscr{P}$ and $L = 35$ POD coefficients $\left(\mathbf{u}_h(\boldsymbol{\mu}), \boldsymbol{\psi}_i\right)_{\mathbb{R}^M}$, $i = 1, \ldots, L$, as teaching inputs.

3.2.3 Nonlinear test case, three parameters

For the last test case concerning the nonlinear Poisson equation, we consider again the stenosis geometry of Fig. 3.11, with $\mu_1, \mu_2 \in [-0.5, 0.5]$ and $\mu_3 \in [1, 5]$. The differential problem we deal with in this section reads:

$$\begin{cases} -\widetilde{\nabla} \cdot \left(\exp(\widetilde{u}(\boldsymbol{\mu})) \, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \right) = \widetilde{s}(\widetilde{x}, \widetilde{y}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{\sigma}_x \sin(\pi \widetilde{\sigma}_x) \cos(\pi \widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu}). \end{cases} \tag{3.11}$$

with the source term $\widetilde{s} = \widetilde{s}(\widetilde{x}, \widetilde{y})$ properly chosen so that the exact solution to the problem is given by $\widetilde{u}_{ex}(\widetilde{x}, \widetilde{y}) = \widetilde{y} \sin(\pi \widetilde{x}) \cos(\pi \widetilde{y})$ for all $\boldsymbol{\mu} \in \mathcal{P}$. Albeit the state equation shows up an exponential nonlinearity and the computational domain presents curvilinear boundaries, both the RB methodologies studied in this work provide accurate solutions, close to the optimal one, i.e., the projection of the truth solution $u_h(\boldsymbol{\mu})$ onto the reduced space $V_{\mathtt{rb}}$. This can be drawn from Fig. 3.17, offering the finite element solution (*top left*) to (3.11) when $\boldsymbol{\mu} = (0.349, -0.413, 4.257)$, and the pointwise errors committed by either the projection onto $V_{\mathtt{rb}}$ (*top right*), or the POD-G method (*bottom left*), or the POD-NN method (*bottom right*). The computational mesh employed consists of 2792 nodes, resulting in $M = 2632$ degrees of freedom (as many as the inner nodes) for the FE scheme. The reduced space $V_{\mathtt{rb}}$ is generated by $L = 30$ basis functions, given by POD of $N = 100$ snapshots. Specifically, the results concerning the POD-NN method have been obtained by employing a neural network equipped with $H_1 = H_2 = 35$ neurons per hidden layer and trained with $N_{tr} = 200$ learning samples. In this way, the POD-NN procedure leads to an error field which shows similar patterns to those featured by the projection error field. This is not surprising, due to the way neural networks are trained within the POD-NN framework (see Section 2.6).

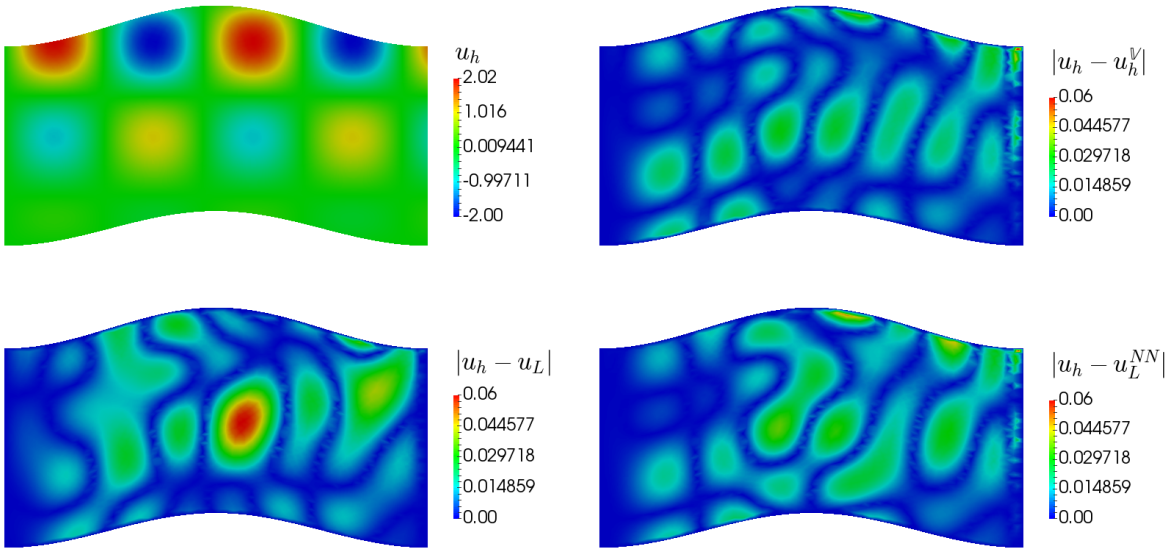


Figure 3.17. FE solution (*top left*) to the Poisson problem (3.11) with $\boldsymbol{\mu} = (0.349, -0.413, 4.257)$, and pointwise errors yielded by either its projection onto $V_{\mathtt{rb}}$ (*top right*), or the POD-G method (*bottom left*), or the POD-NN method (*bottom right*). The results have been obtained by employing $L = 30$ POD modes.

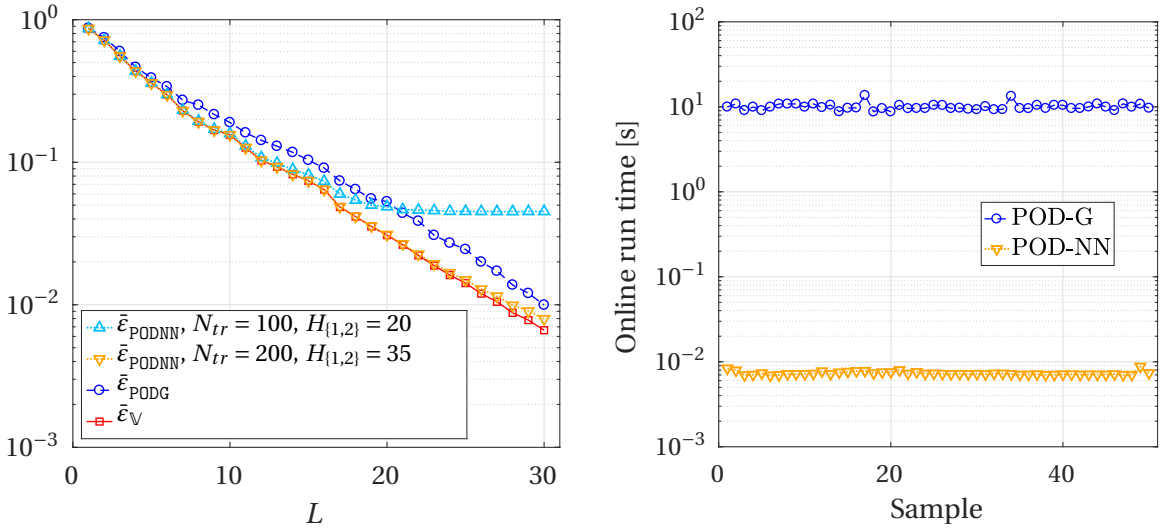


Figure 3.18. Error analysis (*left*) and online CPU time (*right*) for the POD-G and the POD-NN methods applied to problem (3.11). $N_{te} = 50$ randomly picked parameter values are considered. The reduced basis have been generated via POD, relying on $N = 100$ snapshots. The second plot refers to RB models including $L = 30$ modal functions; within the POD-NN framework, a neural network emboding 35 neurons per inner layer has been used.

A quantitative evidence of the efficacy of the RB approximations is given in the plot on the left in Fig. 3.18, which reports a convergence analysis for both the POD-G and the POD-NN methods with respect to the number L of retained modal basis functions. As usual, the error has to be intended as an average over a parameter data set Ξ_{te}, here consisting of $N_{te} = 50$ samples. However, the second plot of Fig. 3.18 reveals how the former scheme produces a response for any online query approximately 10^3 times slowlier than the proposed POD-NN procedure does, preventing any computational gain with respect to the finite element scheme. This assertion can also be extended to the memory demand. In fact, in addition to the assembly of the full-order residual vector $\mathbf{G}_h(\cdot;\boldsymbol{\mu}) \in \mathbb{R}^M$ and the associated Jacobian $\mathbb{J}_h(\cdot;\boldsymbol{\mu}) \in \mathbb{R}^{M\times M}$, the online phase of the POD-G RB method requires the storage of the POD basis $\mathbb{V} \in \mathbb{R}^{M\times L}$. Here, $M = 2632$ and $L = 30$, with the (sparse) Jacobian featuring 18388 non-zero entries. Hence, recalling that a float occupies 4 bytes in memory, the memory footprint approximately amounts to 400 Kb. Instead, the POD-NN method requires only the underpinning neural network to tackle a query. Therefore, assuming 35 computing units within both inner layers, we need a bit more than 9 Kb of RAM, coincident with the space necessary to save the synaptical and bias weights characterizing the network.

Let us now dive deeplier into the analysis of the first plot of Fig. 3.18. Within the POD-NN framework, when the proper orthogonal decomposition of a set of $N = 100$ snapshots is coupled with a three-layers perceptron with 35 neurons per hidden layer, trained to approximate the map (2.71) relying on $N_{tr} = 200$ learning patterns, for each $L \leq 30$ the relative error is smaller than the error committed by the POD-Galerkin procedure. Yet, halfing the number of learning patterns employed, the POD-NN error curve stops decreasing at $L = 20$, then incurring in a plateu. In this regard, Fig. 3.19 suggests that as the size of the available training set decreases, one should coherently reduce the amount of computing units to be embodied in the network, thus to limit the risk of overfitting. This is in agreement with the results previously discussed in this chapter (see, e.g., Fig. 3.5 and Fig. 3.13).

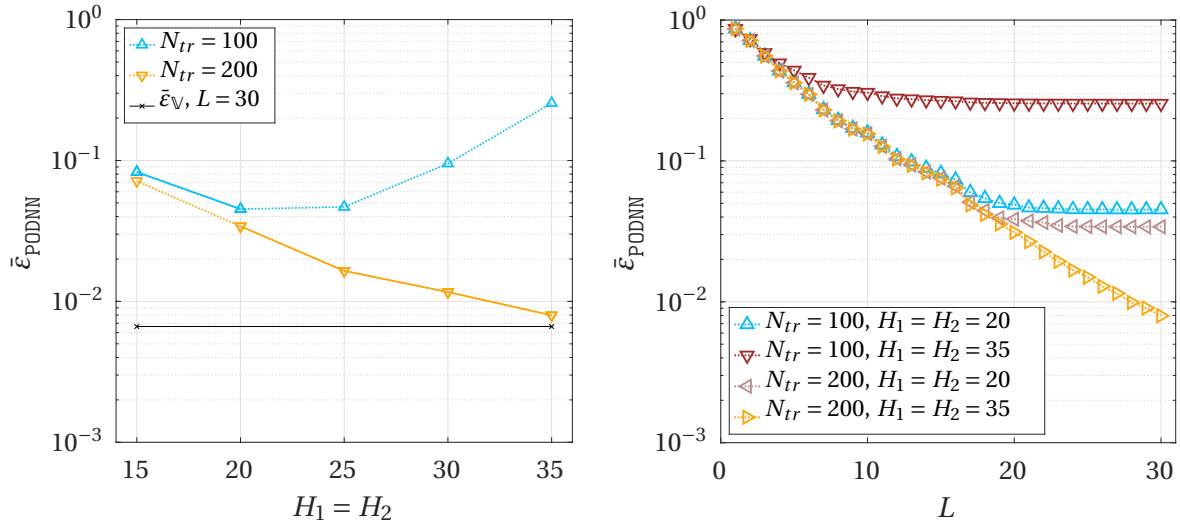


Figure 3.19. Convergence analysis with respect to the number of hidden neurons (*left*) and modal functions (*right*) used within the POD-NN framework applied to problem (3.11). The results provided in the first plot have been obtained using $L = 30$ modes; the solid tracts refer to the steps performed by Algorithm 2.5.

The distinguishing and novel feature of the POD-NN method is represented by the employment of multi-layer perceptrons to recover the coefficients of the reduced model. To motivate this choice, let us pursue a more traditional approach in the interpolation step by resorting to cubic splines [15]. To this end, let $\Xi_\delta \subset \mathscr{P}$ be a tensor-product grid on the parameter domain, based on Chebyshev nodes [2]. In the offline phase, for each $\boldsymbol{\mu}_\delta \in \Xi_\delta$, we compute the truth solution $\mathbf{u}_h(\boldsymbol{\mu}_\delta) \in \mathbb{R}^M$ and we extract the expansion coefficients $\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}_\delta) \in \mathbb{R}^L$. At the online stage, given a new parameter value $\boldsymbol{\mu} \in \mathscr{P}$, the i-th expansion coefficient, $i = 1, \dots, L$, is sought by cubic spline interpolation of the samples

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

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

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

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

For the Poisson problem (3.11), we report in the left plot of Fig. 3.20 the relative errors (averaged on Ξ_{te}) yielded by both the POD-CS and the POD-NN methods. For the former, we provide the results obtained on tensor-product grids consisting of N_δ interpolation points, with $N_\delta \in \{5^3, 7^3, 10^3\}$. For the latter, $H_1 = H_2 = 35$ neurons per hidden layer and

[2] For a given natural number n, the Chebyshev nodes in the interval (a, b) are

$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2n}\pi\right) \quad \text{for } k = 1, \dots, n.$$

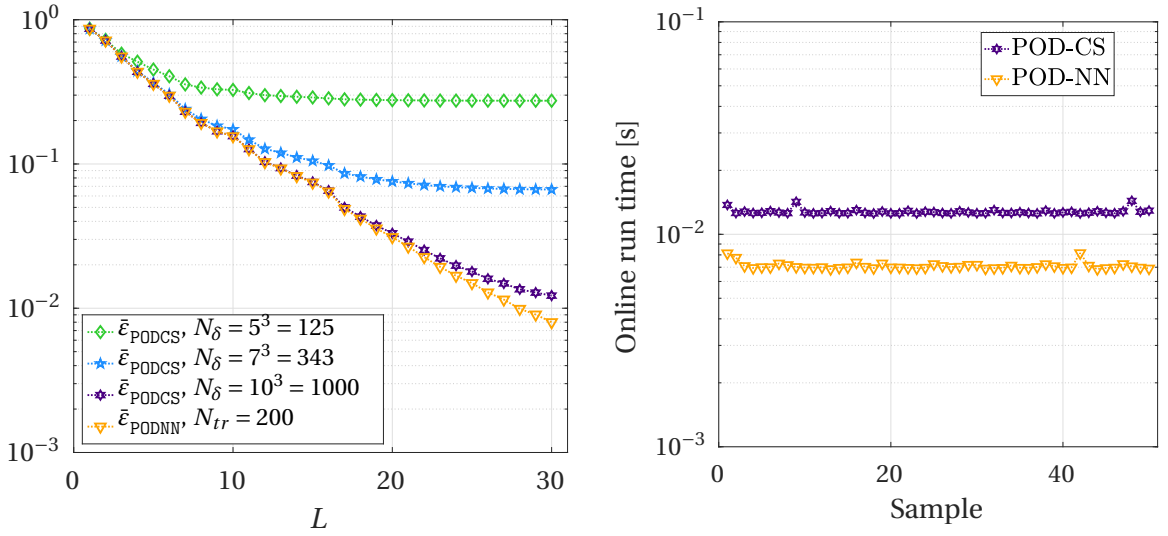


Figure 3.20. Average relative errors (*left*) and online run times (*right*) on Ξ_{te} for the POD-CS and the POD-NN methods applied to problem (3.11). For the latter, the results refer to an MLP equipped with $H_1 = H_2 = 35$ neurons per hidden layer.

$N_{tr} = 200$ training patterns have been used. Although the online performance of the two procedures are basically the same, as shown by the second plot in Fig. 3.20, we observe that the level of predictive accuracy enabled by the POD-NN method can be attained or at least approached by cubic spline interpolation only when this relies on $N_\delta = 1000$ samples. In fact, as mentioned in the Introduction, a standard interpolation technique may require a large number of samples, thus snapshots, to be able to enforce the constraints characterizing the nonlinear manifolds which the reduced bases typically belong to [1]. Hence, although this approach provides a valuable alternative for paremetrized problems with a few parameters, it may be unfeasible in real-life applications involving hundreads of parameters. Conversely, the secret hope behind the choice of a neural network-based regression is that the amount of required snapshots properly scales with the dimension of the parameter space. This would justify the overheads due to the training phase.

3.3 The lid-driven cavity problem

In this section, we address the reduced-basis modeling of the steady uncompressible Navier-Stokes equations (2.9) via the POD-G and the POD-NN procedures. Specifically, we are interested in the numerical simulation of a viscous flow within a parallelogram-shaped cavity, illustrated in Fig. 3.21 (*left*). The geometry of the domain is affected by three parameters: $\mu_1 \in [1, 2]$ and $\mu_2 \in [1, 2]$ define the length of the horizontal and slanting (possibly vertical) edges, respectively, whereas $\mu_3 \in \left[\pi/6, 5\pi/6\right]$ denotes the angle between the oblique sides and the positive $\tilde{x}$-semiaxis. The Dirichlet boundary conditions enforced on the velocity field, graphically represented in Fig. 3.21 (*right*), are such that the flow is driven by a unit horizontal velocity at the top boundary [49]. For this reason, this benchmark is typically referred to as the *lid-driven cavity* problem. In addition, to retain the uniqueness of the solution, we fix the pressure at the lower-left corner [17]. Therefore, to properly

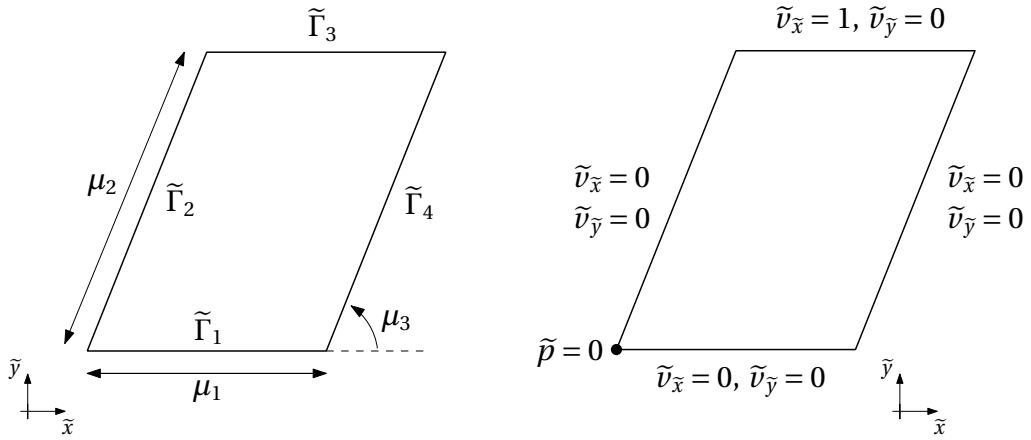


Figure 3.21. Computational domain (*left*) and enforced velocity at the boundaries (*right*) for the lid-driven cavity problem for the uncompressible Navier-Stokes equations.

resolve the steep velocity gradient near the upper boundary and the pressure singularities at the top corners, the computational mesh Ω_h (Fig. 3.22) is refined in the upper part of the domain, leading to $M_{\boldsymbol{v}} = 4868$ degrees of freedom for $\boldsymbol{v}$ and $M_p = 691$ nodal values for p.

Figure 3.23 shows the velocity streamlines (*top*) and the pressure distribution (*bottom*) obtained through the FE method for, from the left to the right, $\boldsymbol{\mu} = \left(1, 2/\sqrt{3}, 2\pi/3\right)$, $\boldsymbol{\mu} = \left(1, 1, \pi/2\right)$ or $\boldsymbol{\mu} = \left(1, 2/\sqrt{3}, \pi/6\right)$. The results refer to a Reynold's number of 400. The Reynold's number is an adimensional group, defined as the ratio between the inertia and the viscous (or friction) force. For the specific problem at hand, it reads:

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

with $\nu = \nu(\boldsymbol{\mu})$ being the dynamic viscosity of the fluid. In our analyses, $\nu(\boldsymbol{\mu})$ is adapted as the geometry varies so that the Reynold's number is either 200 or 400 [3].

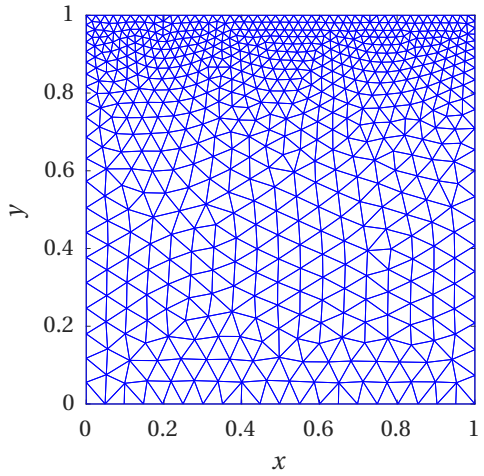
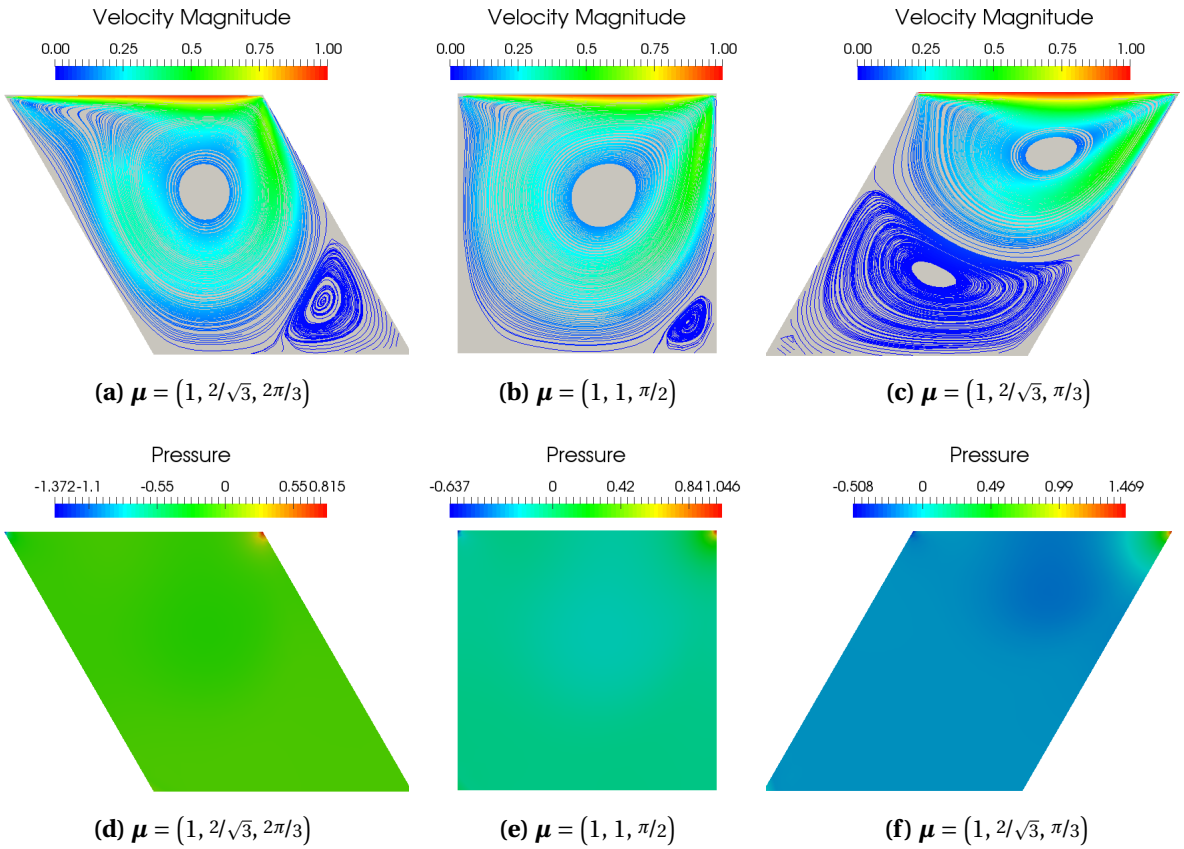


Figure 3.22. The computational mesh Ω_h used in the simulations.

[3] We would like to point out that these values for the Reynold's number have been chosen sufficiently low, so that the flow can be reasonably treated as steady.

As mentioned above, the moving lid induces a large velocity gradient close to the top boundary, while it only slightly affects the fluid in the lower part of the cavity. In turn, this gives rising to two vortices. For $\mu_3 \geq \pi/2$ a large recirculation area expands throughout the domain in the top-down direction, with a smaller vortex, characterized by much lower velocities, appearing in the lower-right corner. By contrast, for $\mu_3 = \pi/6$ the two vortices share the domain almost equally. Actually, if we further decreased the shear angle μ_3, we would note the generation of a third micro-vortex next to the bottom-left corner. In this respect, consider Fig. 3.24, providing the velocity streamlines for the lid-driven cavity benchmark with $\boldsymbol{\mu} = \left(1, \sqrt{2}, \pi/4\right)$ and for a Reynold's number of either 200 (*left*) or 400 (*right*). It is worth noticing that as the Reynold's number decreases, the extent of the upper recirculation area increases, coherently with the augment of the dynamic viscosity.

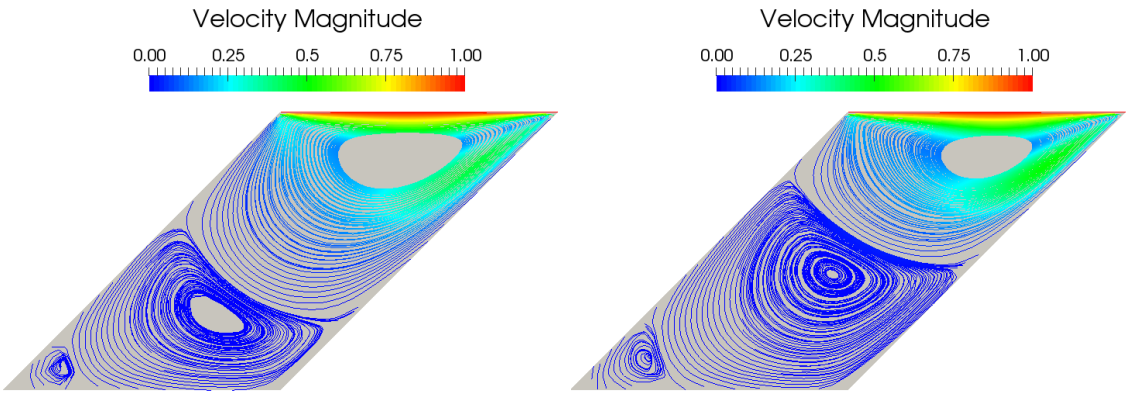
Conversely to the velocity distribution, the pressure field does not undergo remarkable alterations across the parameter space. For all the configurations, a low-pressure region takes place at the upper-left corner of the domain and in the center of the major vortex (although it may be not clearly visible). Instead, the upper-right corner represents a stagnation point for the $\tilde{x}$-velocity, and so therein the pressure assumes larger values than in the rest of the cavity [17].



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

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

Figure 3.23. Velocity streamlines (*top*) and pressure distribution (*bottom*) for the lid-driven cavity problem, computed through the FE method. Three different parameter values are considered; for all configurations, the Reynold's number is 400.



(a) $\boldsymbol{\mu} = \left(1, \sqrt{2}, \pi/4\right)$, $Re = 200$ **(b)** $\boldsymbol{\mu} = \left(1, \sqrt{2}, \pi/4\right)$, $Re = 400$

Figure 3.24. Velocity streamlines for the lid-driven cavity benchmark with $\boldsymbol{\mu} = \left(1, \sqrt{2}, \pi/4\right)$ and either $Re = 200$ (*left*) or $Re = 400$ (*right*). The solutions have been computed through the FE method.

Moving to the reduced-order modeling of the parametrized problem of interest, Fig. 3.25 reports the first 5 modal functions for the velocity $\boldsymbol{v}_h = \boldsymbol{v}_h(\boldsymbol{\mu})$ (*left*), the supremizer solution $\boldsymbol{s}_h = \boldsymbol{s}_h(\boldsymbol{\mu})$ (*center*) and the pressure $p_h = p_h(\boldsymbol{\mu})$ (*right*) for $Re = 200$. These have been respectively obtained via proper orthogonal decomposition of the collections of snapshots $\{\mathbf{v}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^N$, $\{\mathbf{s}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^N$ and $\{\mathbf{p}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^N$, with $\Xi_N = \{\boldsymbol{\mu}^{(i)}\}_{i=1}^N \subset \mathcal{P}$ a training sample generated through latin hypercube sampling. Here, $N = 100$. We recall that each (discrete) supremizer $\mathbf{s}_h(\boldsymbol{\mu}^{(i)})$, $i = 1, \ldots, N$, is related to the corresponding pressure snapshot $\mathbf{p}_h(\boldsymbol{\mu}^{(i)})$ through the linear system (2.66). Then, within the POD-Galerkin RB framework, a reduced-order approximation for the velocity field is searched in the enriched space

$$\overline{V}_{\mathtt{rb}}^{\boldsymbol{v}} = \text{span}\{\psi_1^{\boldsymbol{v}}, \ldots, \psi_{L_{\boldsymbol{v}}}^{\boldsymbol{v}}, \psi_1^{\boldsymbol{s}}, \ldots, \psi_{L_{\boldsymbol{s}}}^{\boldsymbol{s}}\},$$

of dimension $\overline{L}_{\boldsymbol{v}} = L_{\boldsymbol{v}} + L_{\boldsymbol{s}}$, where $\{\psi_i^{\boldsymbol{v}}\}_{i=1}^{L_{\boldsymbol{v}}}$ (respectively, $\{\psi_i^{\boldsymbol{s}}\}_{i=1}^{L_{\boldsymbol{s}}}$) denote the FE functions associated with the first $L_{\boldsymbol{v}}$ (resp., $L_{\boldsymbol{s}}$) left singular vectors of the snapshot matrix $\mathbb{S}_{\boldsymbol{v}}$ (resp., $\mathbb{S}_{\boldsymbol{s}}$). Upon considering a sufficiently large number of supremizer basis functions, the stability at the reduced level follows from Proposition 2.1. In [2], several numerical evidences suggest that a proper value for $L_{\boldsymbol{s}}$ should lay in between $L_p/2$ and L_p, with L_p the number of pressure POD modes included in the RB model. In our simulations, we set $L_{\boldsymbol{s}} = L_p = L_{\boldsymbol{v}}$, leading to a stable reduced (nonlinear) system in $L = 3\,L_{\boldsymbol{v}}$ unknowns.

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

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

Recalling the notation introduced in Section 2.5.3, $\mathbb{V}_{\boldsymbol{v}} \in \mathbb{R}^{M_{\boldsymbol{v}} \times L_{\boldsymbol{v}}}$ (respectively, $\mathbb{V}_p \in \mathbb{R}^{M_p \times L_p}$) gathers the first $L_{\boldsymbol{v}}$ (resp., L_p) left singular vectors of the snapshot matrix $\mathbb{S}_{\boldsymbol{v}}$ (resp., $\mathbb{S}_p$). Moreover, $\mathbf{v}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$ and $\mathbf{p}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$ denote the output vectors provided by two *distinct* multi-layer perceptrons, respectively trained with the ensemble of input-output patterns

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

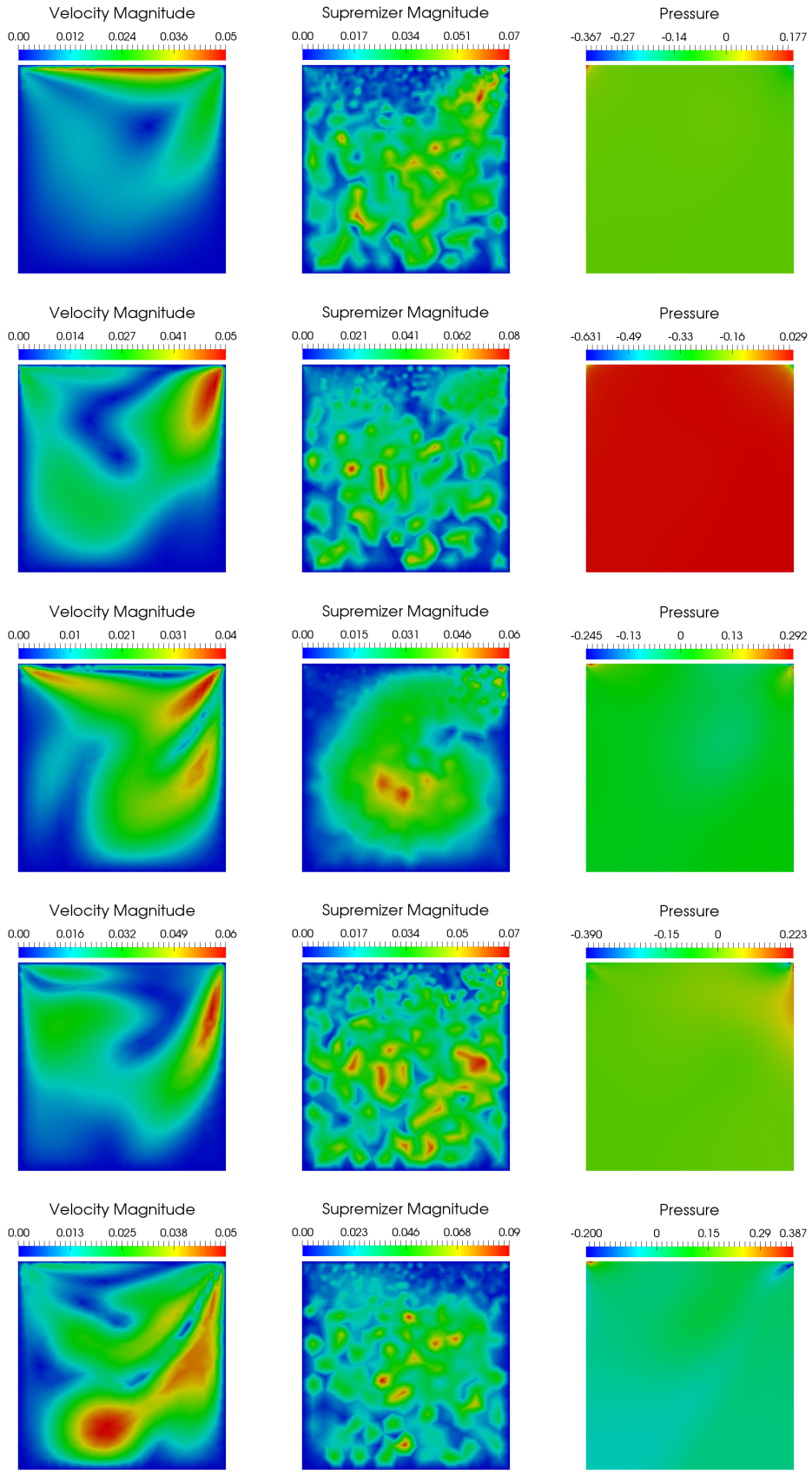


Figure 3.25. First 5 POD modes for the velocity (*left*), supremizers (*center*) and pressure (*right*) for the parametrized lid-driven cavity problem with $Re = 200$.

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

(i) The supremizers $\{\mathbf{s}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^{N_{tr}^p}$ are not involved (either directly or indirectly) in the POD-NN framework. Indeed, the supremizer solutions allow to fulfill an equivalent of the inf-sup condition at the reduced level, thus ensuring the overall stability of the POD-Galerkin procedure (see Eq. (2.67)). However, they do not power-up the accuracy of the method, and so they can be disregarded whenever stability is not an issue.

(ii) The routine resumed by Algorithm 2.5, aiming at finding an optimal network configuration, is applied *separately* to the perceptrons designated to predict the velocity field and to the perceptrons required to approximate the pressure distribution. As a result, once the offline phase of the POD-NN method is completed, we may end up with two completely different networks, equipped with $H_1^{\boldsymbol{\nu}} = H_2^{\boldsymbol{\nu}}$ and $H_1^p = H_2^p$ computing units per hidden layer, respectively.

Figures 3.26 and 3.27 report the error committed by both RB procedures when approximating the velocity field (*left*) and the pressure distribution (*right*) by means of $L_{\boldsymbol{\nu}}$ velocity modes, L_p pressure modes and, exclusively for the POD-Galerkin method, $L_{\boldsymbol{s}}$ supremizer modes, with $5 \leq L_{\boldsymbol{\nu}} = L_{\boldsymbol{s}} = L_p \leq 35$. The former figure refers to a Reynold's number of 200, the latter to a Reynold's number of 400. Please note that for clarity of illustration, the symbols denoting the projection, POD-G and POD-NN errors have been decorated with a superscript (either $\boldsymbol{\nu}$ or p) recalling the state variable they refer to.

While the error yielded on the velocity field shows an almost perfect exponential decay with the number of POD modes included in the RB model, the POD-G method faces difficulties in providing a correct recovery of the pressure, already for $Re = 200$. Indeed, the corresponding error curve may not be monotone, and generally lays at least one order of magnitude above the projection error. On the contrary, the POD-NN method attains

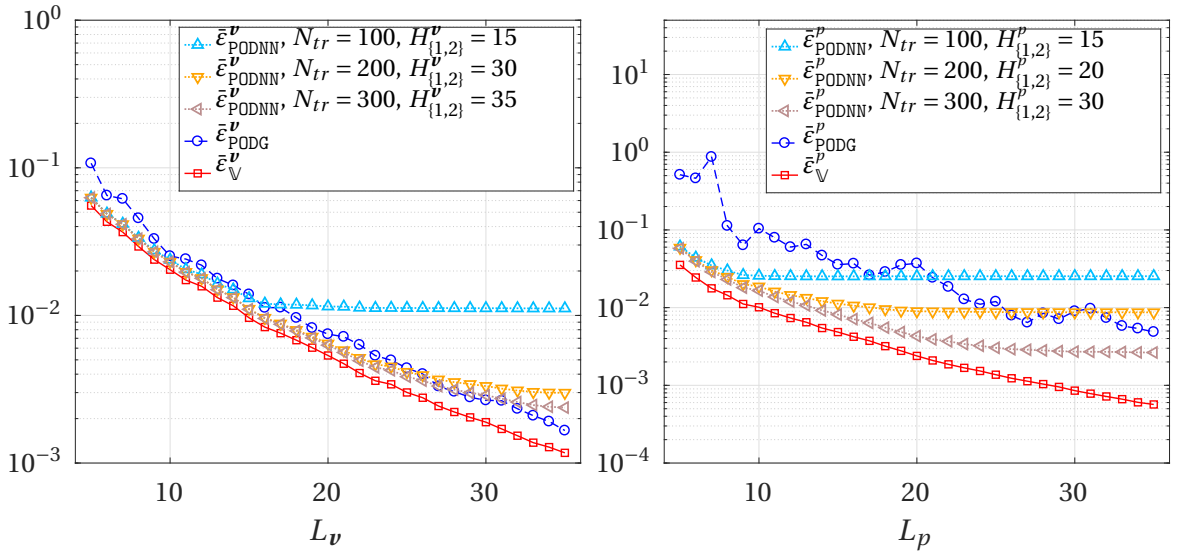


Figure 3.26. Velocity (*left*) and pressure (*right*) error analysis for the POD-G and POD-NN methods applied to the lid-driven cavity problem with $Re = 200$.

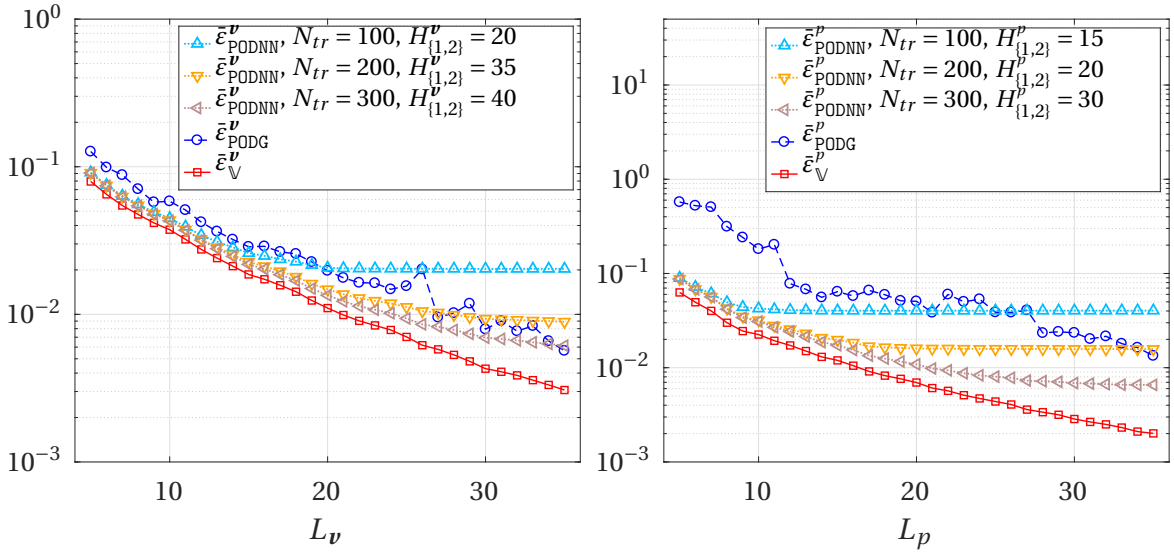


Figure 3.27. Velocity (*left*) and pressure (*right*) error analysis for the POD-G and POD-NN methods applied to the lid-driven cavity problem with $Re = 400$.

a satisfactory predictive accuracy. Particularly, the advantages of resorting to a neural network-based nonlinear regression coupled with a POD procedure are evident when the approximation is built upon a few basis functions, say $L_p < 20$. However, even for $L_p = 35$ (i.e., the dimension of the largest basis tested in our experiments), the POD-NN error is smaller than the POD-G one. Moreover, Algorithm 2.5 recommends to use fewer neurons to predict the POD coefficients for the pressure than for the velocity, thus resulting in a lighter perceptron.

This is confirmed also by Fig. 3.28, which provides a sensitivity analysis of the predictive accuracy featured by the POD-NN method with respect the amount of neurons and training samples used. Observe that for the pressure (*right*), employing more than 30 neurons within each hidden layer is counter-productive, both for $Re = 200$ (*top*) and $Re = 400$ (*bottom*). This assertion agrees with the peculiar role played by the pressure in the parametrized lid-driven cavity problem, with similar patterns featured across the entire parameter domain.

On the contrary, we have seen that the velocity field presents more complex dynamics, highly varying with the domain configuration. As a result, an optimal approximation of the velocity is obtained for the maximum values of $H^{\boldsymbol{\nu}}_i$, $i = 1, 2$, and N_{tr} tested, that is, $H^{\boldsymbol{\nu}}_1 = H^{\boldsymbol{\nu}}_2 = 35$ and $N_{tr} = 300$ for $Re = 200$, $H^{\boldsymbol{\nu}}_1 = H^{\boldsymbol{\nu}}_2 = 40$ and $N_{tr} = 300$ for $Re = 400$. Moreover, we may not be able to exactly attain the same precision enabled by the standard POD-Galerkin procedure, although the results are quite similar. However, this (slight) loss of accuracy is offset by a (great) reduction in the online run time: the POD-NN method takes around 2/100 of seconds per query, against the average 40 seconds required by the POD-G method; see Fig. 3.29. As for the test cases for the Poisson problem discussed in Section 3.2, this comes at the cost of a longer offline phase. In fact, the research and training of sufficiently accurate perceptrons lasted about 4.5 hours here.

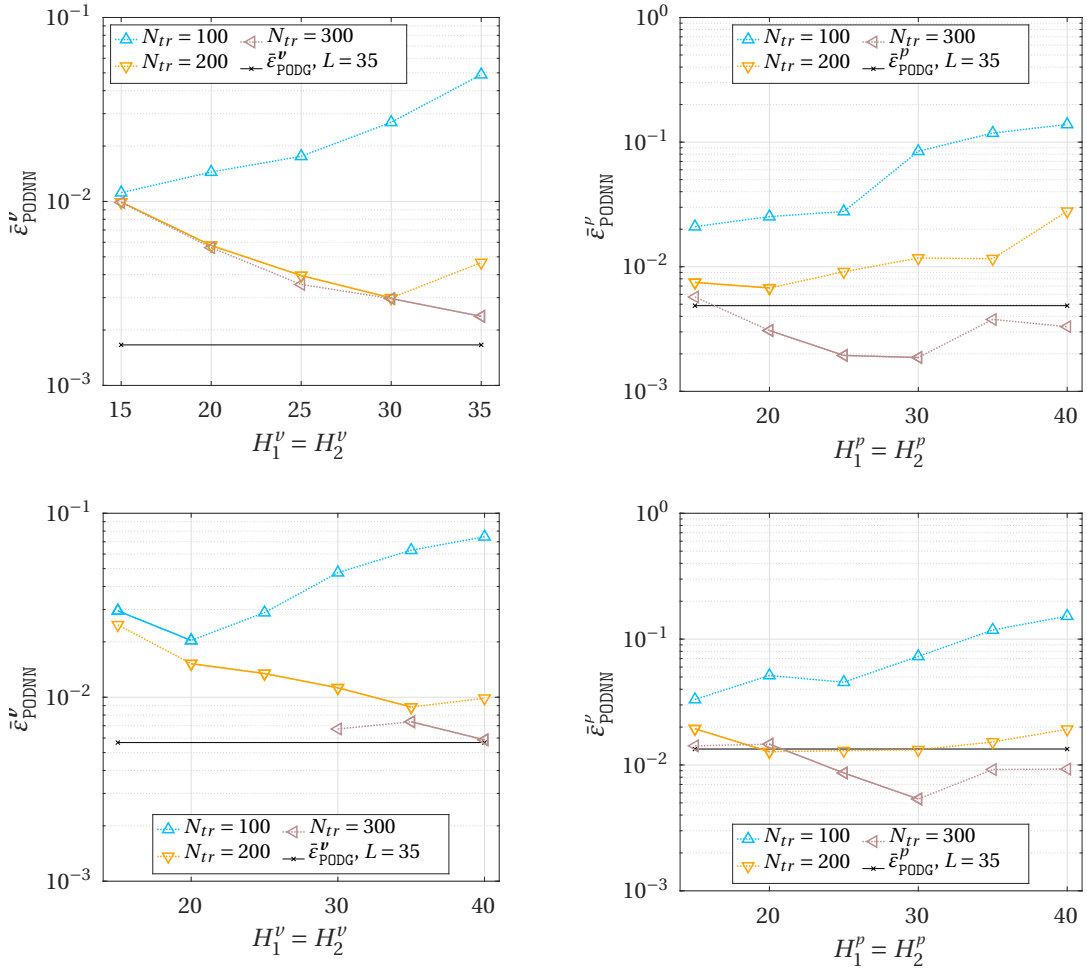


Figure 3.28. Convergence analysis with respect to the number of hidden neurons and training samples used within the POD-NN procedure to approximate the velocity field (*left*) and the pressure distribution (*right*) by means of 35 modal functions. The Reynold's number is either 200 (*top*) or 400 (*bottom*).

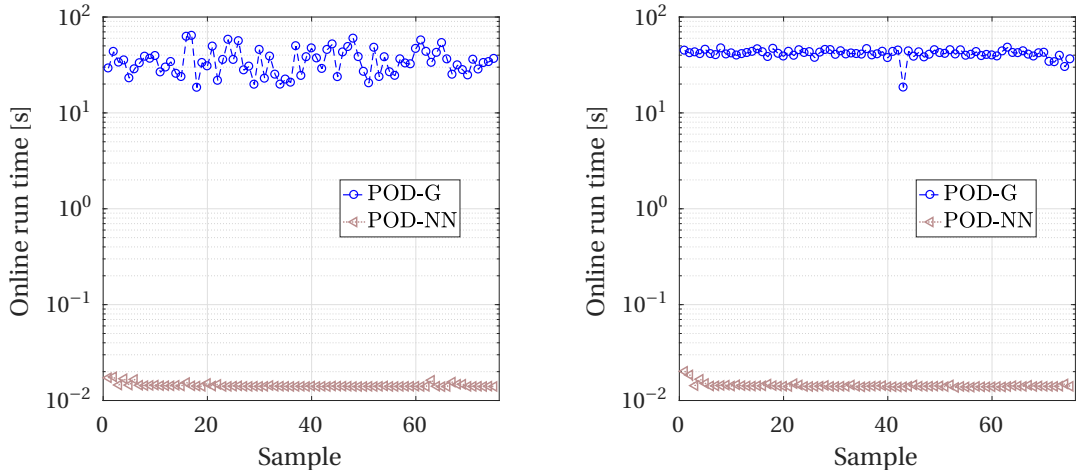
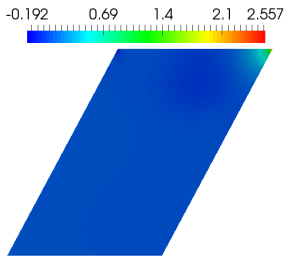
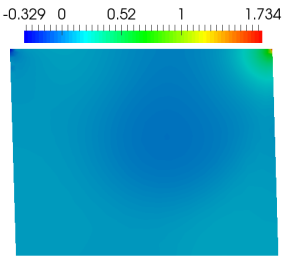


Figure 3.29. Online run times for the POD-G and the POD-NN method applied to the lid-driven cavity problem with $Re = 200$ (*left*) and $Re = 400$ (*right*). $N_{te} = 75$ test configurations are considered. For the POD-NN method, the reported times include the (sequential) evaluation of both neural networks for the velocity and pressure field.

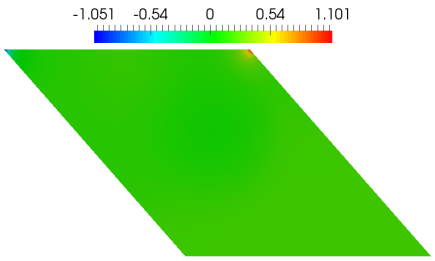
Another numerical evidence of the predictive accuracy of the proposed POD-NN RB method is provided in Fig. 3.30, 3.31 and 3.32, showing the contour plots for the pressure, $\widetilde{x}$-velocity and $\widetilde{y}$-velocity, respectively, computed through the FE (*top row*) and the POD-NN (*bottom row*) scheme. Three different configurations, corresponding to as many input vectors, are considered; the Reynold's number is fixed to 400. We can appreciate the good agreement between the solutions given by the full-order and the reduced-order methods. In particular, for the pressure distribution we also report the solutions provided by the standard POD-G technique. Although the patterns are qualitatively identical to the ones given by the high-fidelity method, with a stagnation point at the upper-left corner and a high-pressure region at the upper-right corner, the colorbars reveal a discrepancy in the maximum and minimum values attained, generally greater than for the POD-NN procedure. This motivates the larger error yielded by the POD-G method.



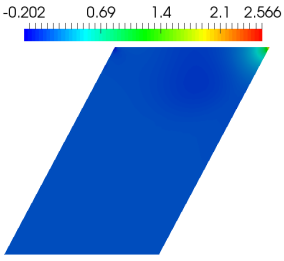
$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), FEM



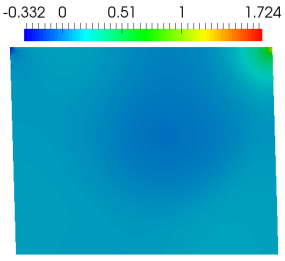
$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), FEM



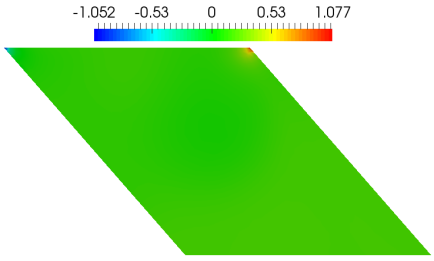
$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), FEM



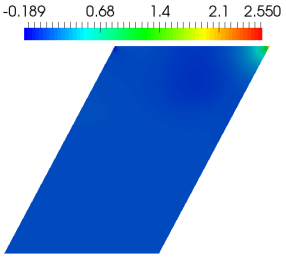
$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), POD-G



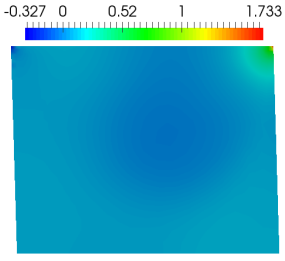
$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), POD-G



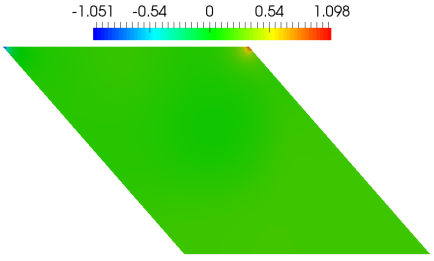
$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), POD-G



$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), POD-NN

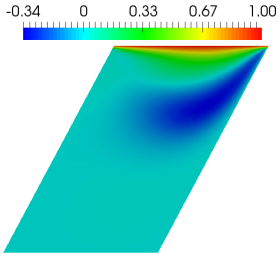


$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), POD-NN

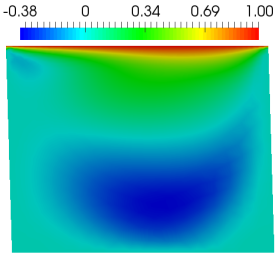


$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), POD-NN

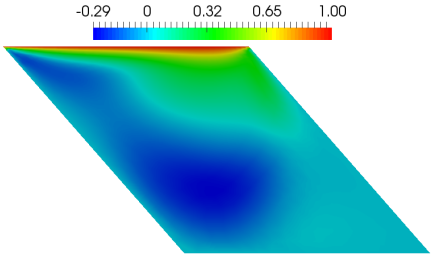
Figure 3.30. Pressure contour at three parameter values, as computed through the FE (*top row*), POD-G (*middle row*) and POD-NN (*bottom row*) method. For each configuration, the Reynold's number is 400.



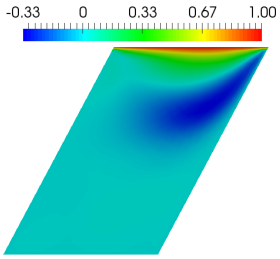
$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), FEM



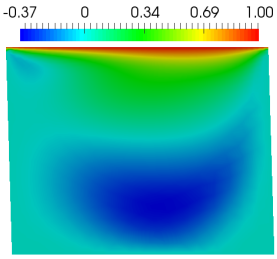
$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), FEM



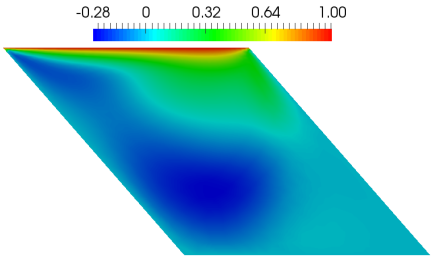
$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), FEM



$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), POD-NN

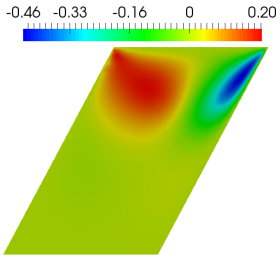


$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), POD-NN

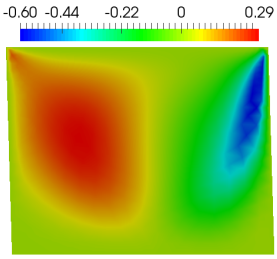


$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), POD-NN

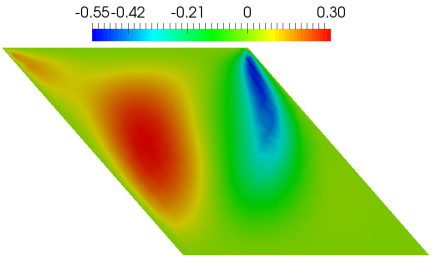
Figure 3.31. $\widetilde{x}$-velocity contour at three parameter values, as computed through the FE (*top row*) and POD-NN (*bottom row*) method. For each configuration, the Reynold's number is 400.



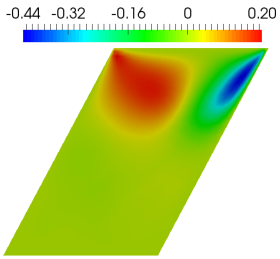
$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), FEM



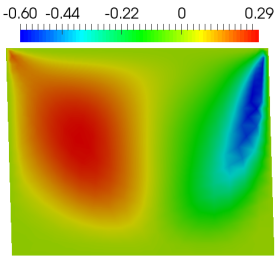
$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), FEM



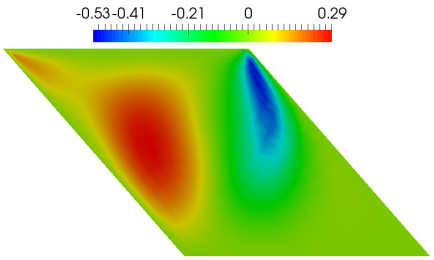
$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), FEM



$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), POD-NN



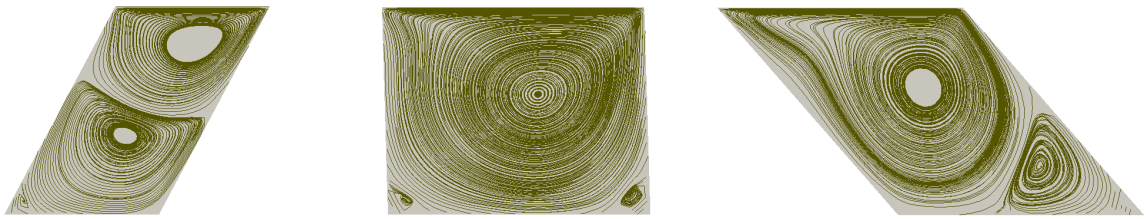
$\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), POD-NN



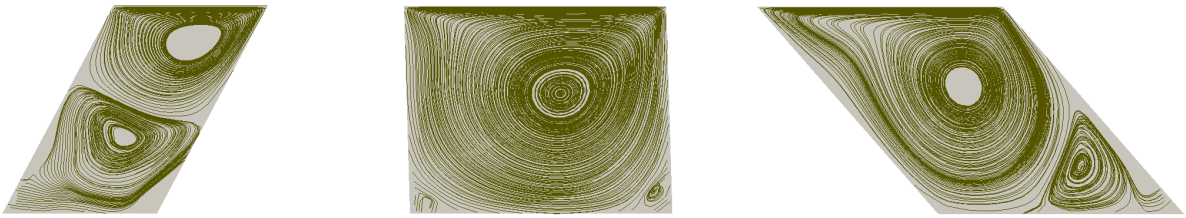
$\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), POD-NN

Figure 3.32. $\widetilde{y}$-velocity contour at three parameter values, as computed through the FE (*top row*) and POD-NN (*bottom row*) method. For each configuration, the Reynold's number is 400.

Lastly, Fig. 3.33 compares the streamlines obtained through the direct method (*top*) and the proposed reduced basis approach (*bottom*) over the three configurations previously considered. Streamlines provide an interesting test bed, as minor variations in velocity contours may lead to substantial differences in the streamlines [12]. However, we observe a good agreement between the two methods. In particular, in the second example, the POD-NN method is still able to detect the two micro recirculation zones at the lower corners of the domain. Whereas, the method partially fails in properly describing the velocity field at the bottom-left corner in the first configuration and at the bottom-right corner in the last configuration. However, these are dead zones, so we can safely disregard these little imprecisions.



$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), FEM $\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), FEM $\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), FEM



$\boldsymbol{\mu}$ = (1.12, 1.70, 1.08), POD-NN $\boldsymbol{\mu}$ = (1.90, 1.50, 1.60), POD-NN $\boldsymbol{\mu}$ = (1.78, 1.99, 2.29), POD-NN

Figure 3.33. Streamlines at three parameter values, as computed through the FE (*top row*) and POD-NN (*bottom row*) method. For each configuration, the Reynold's number is 400.

Conclusion

In this work, we have developed a non-intrusive RB method (referred to as POD-NN) for parametrized time-independent differential problems. The method extracts a reduced basis from a collection of snapshots through a POD procedure and employs multi-layer perceptrons to approximate the coefficients of the reduced model. By exploiting the fundamental results by Cybenko (see (i) and (ii) in Section 1.2.2), we can limit ourselves to perceptrons endowed with two hidden layers. The identification of the optimal number of inner neurons and the minimum amount of training samples which prevents overfitting is performed at the offline stage through an automatic routine (see Algorithm 2.5), relying upon the latin hypercube sampling and the Levenberg-Marquardt training algorithm. On the one hand, this guarantees a complete decoupling between the offline and the online phase, with the latter which entails a computational cost independent of the dimension of the full-order model. On the other hand, this unavoidably extends the offline stage with respect to a standard projection-based RB procedure, making the POD-NN method practically convenient only when the underlying PDE has to be solved for many parameter values (many-query context). At this regard, the method would surely benefit from a GPU implementation, enabling a high parallelization of the code.

The POD-NN method has been successfully tested on the linear and nonlinear Poisson equation in one and two spatial dimensions, and on two-dimensional cavity viscous flows, modeled through the steady uncompressible Navier-Stokes equations. In particular, the proposed RB strategy enabled the same predictive accuracy provided by the POD-Galerkin method while reducing by two or even three order of magnitudes the CPU time required to process an online query. Generally, the optimal network configuration yielded by the developed automatic routine comprised around 35 neurons per hidden layer and required from 200 to 300 training patterns not to incur in overfitting.

All the test cases considered in our numerical studies involved from one up to three parameters, affecting either physical or geometrical factors of the differential problem. In the latter case, we made use of the boundary displacement-dependent transfinite map (see Section 2.2.3) to formulate (and solve) the underlying PDE over a reference, parameter-independent domain. The extension of the POD-NN method to time-dependent problems depending on many parameters is left as future work. In this respect, the secret hope behind a neural network-based interpolation is that the number of training samples, thus snapshots, required to achieve a given level of accuracy properly scales with the dimension of the parameter space. In fact, as observed in Section 3.2.3 already for a three-dimensional parameter domain, this represents the major weak point of traditional interpolation techniques.

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

Acknowledgements / Ringraziamenti

In the first place, let me sincerely thank Prof. Jan Hesthaven: his inspiring ideas, natural positiveness and continuous support throughout the realization of this thesis were invaluable. Moreover, he gave me the opportunity to deal with two effervescent research fields as reduced basis methods and neural networks.

I wish to express my gratitude to Prof. Paolo Zunino, who accepted the role of thesis co-supervisor without even knowing me. I really appreciated his kindness and availability.

I must also thank Prof. Luca Dedè, thesis external examiner, for the time and effort he will invest in reviewing this dissertation.

I am grateful to Caterina, for the time she spent reading this work and for her valuable comments, which really improved the quality of the thesis.

I would also like to thank Anne-Dominique, for the warmth and kindness with which she welcomed me, making me feel at home.

Ringrazio i *Bergamo scende Lecce* (Roberta, Giorgio, Luca, Jessica, Sofia, Fabio, Giorgia, Lorenzo), con cui ho condiviso molti bei momenti che porto nel mio cuore. Mai avrei pensato di trovare amici come voi all'università - mi sono dovuto ricredere.

Ringrazio anche il *Gruppo Losanna* (Ondine, Marco, Francesco, Nicolò, Stefano), Luca e Davide per gli ultimi tre anni a cavallo tra Milano e Losanna. Anche se in certi momenti la fine sembrava irraggiungibile, è stata comunque un'esperienza indimenticabile, e sicuramente buona parte del merito va anche a voi. Vorrei ringraziare in modo particolare (non me ne vogliano gli altri) Marco, con cui mi sono trovato alla perfezione in più di un progetto, e Nicolino, con cui ho piacevolmente condiviso l'ufficio negli ultimi mesi.

Un grazie veramente di cuore alla mia famiglia, in particolare ai miei genitori, per avermi supportato emotivamente ed economicamente. Senza di voi questo traguardo sarebbe stato irraggiungibile.

Ed infine, un ringraziamento particolare a Daniela, per non avermi mai fatto pesare la distanza che ci separava, supportandomi e sopportandomi sempre.

Stefano Ubbiali
Lausanne, July 3rd, 2017

Bibliography

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

[19] 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.

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

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

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

[23] 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`.

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

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

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

[27] 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.

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

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

[30] 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.

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

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

[33] 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.

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

[35] 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.

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

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

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

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

[40] 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.

[41] 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.

[42] 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`.

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

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

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

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

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

[48] 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.

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

[50] 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.

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

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

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

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

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

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

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

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

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