Politecnico di Milano
Dipartimento di Matematica "F. Brioschi"

École Polytechnique Fédérale de Lausanne
Chair of Computational Mathematics and Simulation Science

Master of Science in
Computational Science and Engineering

Reduced order modeling of nonlinear problems using neural networks

Candidate
Stefano Ubbiali

Supervisors
Prof. Jan S. Hesthaven
Prof. Matteo Zunino

A.Y. 2016 - 2017

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction to 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., [13, 15, 26], from which we retrieved many of the informations provided in this chapter.

Throughout this work we confine the attention to the most-spread 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. Actually, in the numerical experiments we carried out and whose results will be discussed in Chapter **??**, we mainly refer to a variant of backpropagation - the Levenberg-Marquardt algorithm.

Before moving to the description of technical neural networks, let us provide a brief excursus on 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 over into a mathematical, simplified 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 [15].

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 each other by means of electric signals, traveling through a thick net of

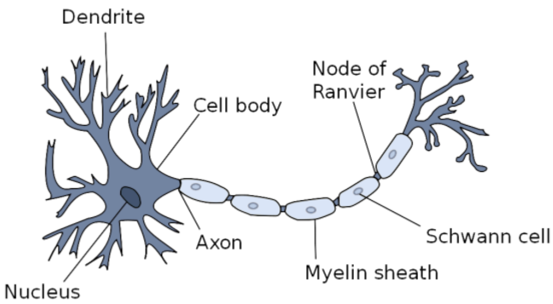

Figure 1.1. Simplified representation of a biological neuron, with the components discussed in the text; retrieved from [26].

connections. For instance, in a human being each neuron is linked in average with $10^3 - 10^4$ other neurons. 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 [26].

From a simplifying 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 [43]. 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). However, this insulating film is not continuous, rather presents gaps at regular intervals called *nodes of Ranvier*, which lets the signal 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. On the other hand, neurotransmitters do not simply broadcast the action potential. Indeed, we can distinguish between excitatory and inhibitory neurotransmitters, respectively amplifying or modulating

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

Different studies have unveiled the tight correlation between the synapses the neurons establish among each other, and the tasks a neural network can address [13]. That is, the set of interneuron connection strengths represent the information storage, i.e., the knowledge, of a neural network [26]. 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, 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 successfull training a neural network should also show a discrete level of *fault tolerance* against external errors, e.g. noisy inputs. It worths notice here that the nervous system is also naturally fault tolerant against *internal* errors. Indeed, in case a neuron or a (relatively small) group of neurons got damaged or died, the other processing nodes would take care of its tasks, so that the overall cognitive capabilities would be only slightly affected [26].

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 [33]. Exactly as its original counterpart, it consists of a collection of processing units, called (artificial) neurons, and directed weighted synaptic connections between the neurons themselves. Data travel among neurons through the connections, following the direction imposed by the synapses themselves. 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 [43].

Formally, a neural network could be defined as follows [26].

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* $w : \mathcal{V} \rightarrow \mathbb{R}$ *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.

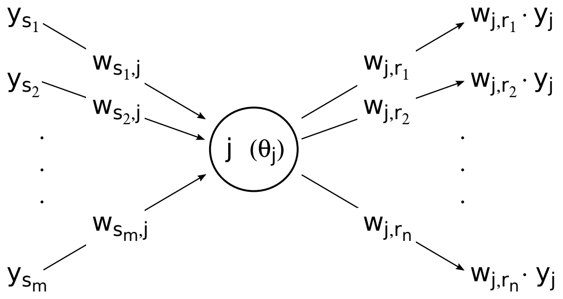

Figure 1.2. Visualization of the generic j-th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j}\ y_{s_1}, \dots, w_{s_m,j}\ y_{s_m}\}$ coming from the sending neurons $s_1, \dots, s_m$, and fires y_j, sent to the target neurons $\{r_1, \dots, r_n\}$ through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

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 Figure 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 $t + \Delta t$. 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; on the other hand, it dramatically eases 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 hereunder in the same order they get involved in the data flow.

Propagation function. The propagation function f_{prop} converts the vectorial input $\boldsymbol{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ into a scalar u_j often called *net input*, i.e.,

$$u_j = f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) . \tag{1.1}$$

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:

$$f_{prop}(w_{s_1,j}, \dots, w_{s_m,j}, y_{s_1}, \dots, y_{s_m}) = \sum_{k=1}^{m} w_{s_k,j}\ y_{s_k} . \tag{1.2}$$

The function (1.2) 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 shortly referred to as *activation*, quantifies at 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 $\theta_j \in \mathbb{R}$ [26]:

$$a_j = f_{act}(u_j; \theta_j) = f_{act}(\sum_{k=1}^{m} w_{s_k,j}\ y_{s_k}; \theta_j) , \tag{1.3}$$

where we have employed the weighted sum (1.2) 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 $|f'_{act}|$ of the derivative of the activation function is maximum. Then, the activation function reacts particularly sensitive when the net input u_j hits the threshold value θ_j [26].
Furthermore, noting that θ_j is a parameter of the network, one may like to adapt it through a training process, exactly as can be done for the synaptic weights, as we shall see in Section 1.2.3. However, θ_j is currently incorporated in the activation function, making its runtime access somehow cumbersome. This is typically overcome by introducing a *bias neuron* in the network. A bias neuron is a continuously firing neuron, with constant output $y_b = 1$, which gets directly connected with neuron j, assigning the *bias weight* $w_{b,j} = -\theta_j$ to the connection. As can be deduced by Figure 1.3, θ_j is now treated as a synaptic weight, while the neuron threshold is set to zero. Moreover, the net input becomes

$$u_j = \sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} - \theta_j\,, \tag{1.4}$$

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

$$a_j = f_{act}(\sum_{k=1}^{m} w_{s_k,j}\, y_{s_k} - \theta_j)\,. \tag{1.5}$$

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: just 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 shall 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, as the Heaviside or binary function, which assumes only 0 or 1, according to whether the argument is negative or positive, respectively:

$$f_{act}(v) = \begin{cases} 0\,, & \text{if } v < 0, \\ 1\,, & \text{if } v \geq 0. \end{cases} \tag{1.6}$$

Neurons characterized by such an activation function are usually named McCulloch-Pitts neurons, after the seminal work of McCulloch and Pitts [15], and their employment is usually limited to single-layer perceptrons implementing boolean logic (see Section 1.2.2). In addition, note that (1.6) is discontinuous, with a vanishing derivative everywhere except that in the origin, thus not admissible for the backpropagation training algorithm presented 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 [15]. Sigmoid functions are s-shaped and monotically increasing, and assume values in a bounded interval, typically $[0,1]$, as the logistic function,

$$f_{act}(v) = \frac{1}{1+e^{-v/T}} \quad \text{with } T > 0\,, \tag{1.7}$$

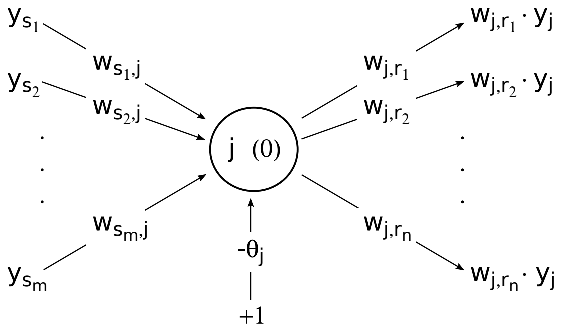

Figure 1.3. Visualization of the generic j-th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j}\ y_{s_1}, \dots, w_{s_m,j}\ y_{s_m}, -\theta_j\}$ coming from the sending neurons $s_1, \dots, s_m$, b, respectively, with b the bias neuron. The neuron output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

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

$$f_{act}(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}} . \tag{1.8}$$

Both functions are displayed in Figure 1.4. Note that the logistic function resemble the Heaviside function as T decreases.

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

$$y_j = f_{out}(a_j) ; \tag{1.9}$$

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 $\boldsymbol{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ of the neuron is generally vectorial, i.e., $m > 1$, the output is scalar. The output y_j could then either be sent to other neurons, included 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 worths mention 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 actually 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 distinguish *source* neurons, supplying the network with the respective components of the activation pattern (input vector) [15]. The role of source neurons will be clearer in the next section, where we will introduce the multilayer feedforward neural network. Here we just mention that such a neuron receives a scalar and unweighted input, which is simply forward to the connected neurons; no computations are performed.

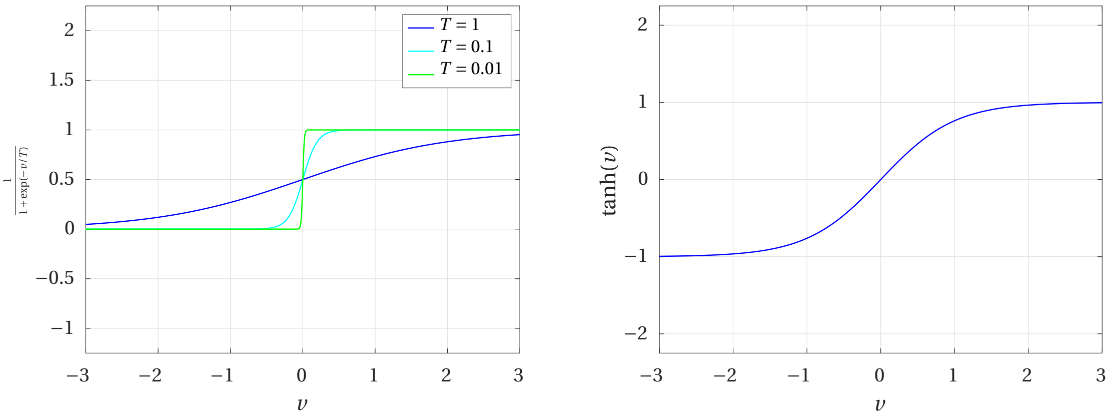

Figure 1.4. Left: logistic function (1.7) for three values of the parameter T; note that as T decreases, the logistic function resemble the Heaviside function. Right: hyperbolic tangent.

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 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* [41], 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, $k = 1, \ldots, K$, and an *output layer* of M_O computing neurons. As characteristic property, neurons in a layer can only be connected with neurons in the next layer towards the output layer. Then, an *activation pattern* $\boldsymbol{p} \in \mathbb{R}^{M_I}$, 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 signal gives 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 $\boldsymbol{q} \in \mathbb{R}^{M_O}$ of the network, response to the input pattern $\boldsymbol{p}$[2]. Hence, a feedforward network establish 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 neurons they present, or, equivalently, the number of layers of trainable weights. Single-layer perceptrons (SLPs) just consist of the input and output layer; no hidden layers. Note that the layer which the name refers to is the output layer; the input layer is not accounted for since it does not perform any calculation. Despite their quite simple structure, the range of application of single-layer perceptrons is rather limited. Indeed, consider a binary input vector, supplied to an SLP provide with a unique output neuron, equipped with a binary

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

activation function. The 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 [26]. On the other hand, the share of linearly separable sets decreases as the space dimension increases, making single-layer perceptrons seldom attractive.

Conversely, multi-layer perceptrons (MLPs), providing at least one hidden layer, are universal function approximators, as stated by Cybenko [7, 8]. In detail:

(i) a multi-layer perceptron with one layer of *hidden neurons* and differentiable activation functions can approximate any *continuous* function [8];

(ii) a multi-layer perceptron with two layers of *hidden neurons* and differentiable activation functions can approximate *any function* [7].

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 by adding one hidden layer, and any arbitrary set by adding two hidden layers; further increasing the number of hidden neurons would not improve the representation capability of the network. However, we should point out that (i) and (ii) do not give any practical advice on both the number of hidden neurons and the number of samples required to teach the network.

An instance of a three-layer (i.e., two hidden layer plus the output layer) feedforward network is offered in Figure **??**. In this case, we have $M_I = 3$ input neurons (denoted with the letter i), $H_1 = H_2 = 6$ hidden neurons (letter h), and $M_O = 4$ output neurons (letter o). In particular, we point out that 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 can 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 neither input or output neurons any more: neurons are all equivalent. Then, 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. So, 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 [20], inspired by the behaviour of electrically charged particles within a magnetic field, and the self-organizing maps by Kohonen [25], 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 $\boldsymbol{\phi}$ between the parameters $\boldsymbol{\mu}$ of a parametrized partial differential equation, and the coefficients $\boldsymbol{\alpha}$ of the corresponding reduced basis solution. To accomplish this task, we rely on a collection of samples $\{\boldsymbol{\mu}_i, \boldsymbol{\alpha}_i\}_{i=1}^{N_{tr}}$ generated through a high-fidelity model. Although a detailed explanation will be provided later in Chapter **??**, what is worth notice here is that we are concerned with a *continuous function approximation* problem. Then, motivated by what previously said in the section, a multilayer feedforward neural network turns out as the most suitable network architecture for our purposes.

We are now left to investigate the way the weights of a perceptron can be *trained* to meet our purposes.

1.2.3 Training a multilayer feedforward neural network

As widely highlighted so far, the primary characteristic of a neural network lies in the capability to *learn* from the environment, 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, a learning procedure is an *algorithm*, typically iterative, modifying the neural network parameters to make it knowledgeable of the specific environment it operates in [15]. Specifically, this entails that after a successfull training, the neural network has to provide reasonable responses for unknown problems of the same class the network was acquainted with during training. This property is known as *generalization* [26].

Training algorithms can be firstly 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) $\{\boldsymbol{p}_i\}_{i=1}^{N_{tr}}$, and corresponding desired responses $\{\boldsymbol{t}_i\}_{i=1}^{N_{tr}}$, called *teaching inputs*. Then, $\boldsymbol{t}_i$ is the output the neural network should desirably provide when it gets fed with $\boldsymbol{p}_i$. As we shall see, any training procedure aims to minimize (in some appropriate norm) the discrepancy between the *desired* output $\boldsymbol{t}_i$ and the *actual* output $\boldsymbol{q}_i$ given by the network as response to $\boldsymbol{p}_i$.

Unsupervised learning. Although supervised learning is a simple and intuitive paradigm, there exist many tasks which require a different approach to be tackled. Consider 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, a training pattern just consist of an input pattern. Since no desired 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 saying whether the network is wrong or right. In the former case, the *feedback* or *reward* may also indicate to which extent the network is wrong [26]. 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

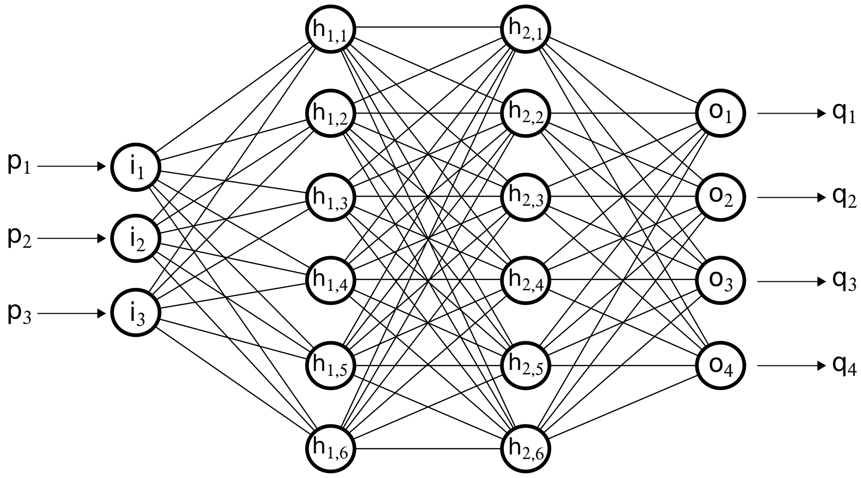

Figure 1.5. A three-layer feedforward neural network, with 3 input neurons, two hidden layers each one consisting of 6 neurons, and 4 output neurons. Within each connection, information flows from left to right.

between a long-term and a short-term reward [23].

Clearly, the choice of the learning paradigm is task-dependent. In particular, function approximation (i.e., what we are interested in) perfectly fits the *supervised learning* paradigm. Indeed, consider the nonlinear unknown function $\boldsymbol{f}$,

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

and a set of labeled examples $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N_{tr}}$. The task implies to approximate $\boldsymbol{f}$ over a domain $V \subset \mathbb{R}^{M_I}$ up to a user-defined tolerance ϵ, i.e.,

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

where $\boldsymbol{F} : \mathbb{R}^{M_I} \to \mathbb{R}^{M_O}$ is the actual input-output map established by the neural network, and $|| \cdot ||$ is some suitable norm on $\mathbb{R}^{M_O}$. Surely, as necessary condition the neural system must satisfy to well approximate $\boldsymbol{f}$ for each input in the domain V, the system should provide accurate predictions for the input patterns, i.e.,

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

Then, we could train the network through a supervised learning algorithm, employing $\{\boldsymbol{x}_i\}_{i=1}^{N_{tr}}$ as input patterns and $\{\boldsymbol{y}_i\}_{i=1}^{N_{tr}}$ as teaching inputs. That is,

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

As defined in the first part of the 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 literature, sometimes tailored on specific applications. Nevertheless, most of them relies on the well-known Hebbian rule, proposed by Donal O. Hebb in 1949 [16]. Inspired by neurobiological considerations, the rule can be stated in a two-steps fashion [15]:

(i) if two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously) 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 [15]. 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

$$\Delta w_{i,j}(t) \sim \eta \; y_i(t) \; a_j(t) \, , \tag{1.10}$$

where $\eta > 0$ is the *learning rate*, and we have 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

$$w_{i,j}(t+1) = w_{i,j}(t) + \Delta w_{i,j}(t) \, . \tag{1.11}$$

Many of the supervised learning rules proposed in literature, included the backpropagation of error derived later, can be recast in the following form, which turns out as a generalization of the Hebbian rule (1.10) [26]:

$$\Delta w_{i,j} = \eta \; h(y_i, w_{i,j}) \; g(a_j, t_j) \, , \tag{1.12}$$

with the functions h and g dependent on the specific learning rule, and t_j the j-th component of the teaching input $\boldsymbol{t}$. Note that all variables involved in (1.12) 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 (1.12) 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 somehow measures the discrepancy between the neural network knowledge of the surrounding environment, and the actual state of the environment itself. In other terms, the larger the performance function, the farer the neural network representation of the world is from the actual reality, i.e., the farer we are from the application goal. Therefore, every learning rule aims to *minimize* the performance E as much as possible. For this purpose, E should be intended as a scalar function of the free parameters, i.e., the weights, of the network, namely

$$E = E(\boldsymbol{w}) \in \mathbb{R} \, . \tag{1.13}$$

Recalling the notation and assumptions introduced in Definiton 1.1, here $\boldsymbol{w} \in \mathbb{R}^{|\mathcal{V}|}$ is a vector collecting the weights $\{w_{i,j} = w((i,j))\}_{(i,j)\in\mathcal{V}}$, with $\mathcal{V}$ the set of admissible connections in the network[3]. Thus, Equation (1.13) implies that the point over the error surface reached at the end of a successful training process provides the *optimal* configuration $\boldsymbol{w}_{opt}$ for the network.

The steps a generic supervised training procedure should pursued 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, 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 only after the network has been fed with all input patterns and the corresponding errors have been accumulated. Therefore, we should distinguish between the *specific error* $E_{\boldsymbol{p}}(\boldsymbol{w})$, specific to the activation pattern $\boldsymbol{p}$, and the *total error* $E(\boldsymbol{w})$ accounting for all specific errors, namely

$$E(\boldsymbol{w}) = \sum_{\boldsymbol{p} \in P} E_{\boldsymbol{p}}(\boldsymbol{w}) \, , \tag{1.14}$$

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

$$E_{\boldsymbol{p}}(\boldsymbol{w}) = \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\boldsymbol{p},j} - q_{\boldsymbol{p},j} \right)^2 \, , \tag{1.15}$$

where we have added the subscript $\boldsymbol{p}$ to the components of the teaching input $\boldsymbol{t}$ and the actual output $\boldsymbol{q}$ to remark they refer to the input pattern $\boldsymbol{p}$. Accordingly, we could provide the following definition for the accumulated MSE:

$$E(\boldsymbol{w}) = \sum_{\boldsymbol{p} \in P} E_{\boldsymbol{p}}(\boldsymbol{w}) = \sum_{\boldsymbol{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\boldsymbol{p}_i,j} - q_{\boldsymbol{p}_i,j} \right)^2 \, . \tag{1.16}$$

So far, we have not yet discussed how the update $\Delta w_{i,j}$ for the weight $w_{i,j}$ can be practically computed at each iteration. That is, we have still to rigourously define the functions h and g involved in the generalized Hebbian rule (1.12). Then, let us recall that any given operation carried out by the neural network can be thought as a point over the error surface $E(\boldsymbol{w})$. Therefore, to increase the performance of the network, we need to iteratively move toward a (local) minimum of the surface [15]. For this purpose, we may employ a *steepest descent* technique, thus following the direction of the anti-gradient, i.e.,

$$\Delta w_{i,j} = -\eta \frac{\partial E(\boldsymbol{w})}{\partial w_{i,j}} \, , \tag{1.17}$$

$\eta > 0$ being the learning rate, modulating the size of the step; its role will be clearer later in the chapter. Among the others, the *backpropagation of error* [32] 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.

[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, then such connections will not be included in $\mathcal{V}$. In this way, we reduce the size of $\boldsymbol{w}$ - which is a practical advantage.

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

Algorithm 1.1 Backbone of any supervised online learning algorithm; note that the full procedure ends when all training patterns yield an error which is below a defined threshold.

```
Input: neural network $(\mathcal{N}, \mathcal{V}, \boldsymbol{w}_0)$, training set $P = \{\boldsymbol{p}_i, \boldsymbol{t}_i\}_{i=1}^{N_{tr}}$,
       metric $d(\cdot,\cdot) : \mathbb{R}^{M_O} \times \mathbb{R}^{M_O} \to \mathbb{R}$, tolerance $\epsilon$, maximum number of epochs $T$
Output: trained neural network $(\mathcal{N}, \mathcal{V}, \boldsymbol{w}_{opt})$
$t = 0$, $i = 1$, $k = 0$
$\boldsymbol{w}(0) = \boldsymbol{w}_0$
while $t < T$ and $k < N$ do
    evaluate output vector $\boldsymbol{y}_{\boldsymbol{p}_i}(t)$, correspondent to input pattern $\boldsymbol{p}_i$
    $E_{\boldsymbol{p}_i}(\boldsymbol{w}(t)) = d(\boldsymbol{y}_{\boldsymbol{p}_i}(t), \boldsymbol{t}_1)$
    if $E_{\boldsymbol{p}_i}(\boldsymbol{w}(t)) < \epsilon$ then
        $k \leftarrow k + 1$
    else
        $k = 0$
        compute weight update $\Delta\boldsymbol{w}(t)$ based on $E_{\boldsymbol{p}_i}(\boldsymbol{w}(t))$
        $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta\boldsymbol{w}(t)$
    end if
    $t \leftarrow t + 1$, $i \leftarrow i \pmod{N} + 1$
end while
$\boldsymbol{w}_{opt} = \boldsymbol{w}(t)$
```

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 widely exploiting the chain rule, the backpropagation of error provides an operative formula for the evaluation of the antigradient of the error function $E(\boldsymbol{w})$ in an arbitrary point $\boldsymbol{w}$, thus allowing to compute the update $\Delta w_{i,j}$ for the weight $w_{i,j}$ according to (1.17). For this purpose, we shall need to distinguish whether the successor neuron j is either an output or an inner neuron. To improve the clarity of the following mathematical derivation, we shall decorate any variable concerning a neuron with the subscript $\boldsymbol{p}$, thus to explicitly specify the input pattern $\boldsymbol{p}$ we refer to.

Let $S = \{s_1, \ldots, s_m\}$ the set of m neurons sending their output to j through the synapses $\{(s_1, j), \ldots, (s_m, j)\}$. Recalling the definition (1.14) of the accumulated error, via the chain rule we can formulate the derivative of $E(\boldsymbol{w})$ with respect to the weight $w_{i,j}$ as follows:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_{i,j}} = \sum_{\boldsymbol{p} \in P} \frac{\partial E_{\boldsymbol{p}}(\boldsymbol{w})}{\partial w_{i,j}} = \sum_{\boldsymbol{p} \in P} \frac{\partial E_{\boldsymbol{p}}(\boldsymbol{w})}{\partial u_{\boldsymbol{p},j}} \frac{\partial u_{\boldsymbol{p},j}}{\partial w_{i,j}}. \tag{1.18}$$

Let us focus on the product

$$\frac{\partial E_{\boldsymbol{p}}(\boldsymbol{w})}{\partial u_{\boldsymbol{p},j}} \frac{\partial u_{\boldsymbol{p},j}}{\partial w_{i,j}} \tag{1.19}$$

occurring in Equation (1.18). Assuming the propagation function be represented as the

Algorithm 1.2 Backbone of any supervised offline learning algorithm; the procedure to compute the accumulated error is provided as well.

```
Input:  neural network $(\mathscr{N}, \mathscr{V}, \boldsymbol{w}_0)$, training set $P = \{\boldsymbol{p}_i, \boldsymbol{t}_i\}_{i=1}^{N_{tr}}$,
        metric $d(\cdot,\cdot) : \mathbb{R}^{M_O} \times \mathbb{R}^{M_O} \rightarrow \mathbb{R}$, tolerance $\epsilon$, maximum number of epochs $T$
Output: trained neural network $\left(\mathscr{N}, \mathscr{V}, \boldsymbol{w}_{opt}\right)$

$t = 0$
$\boldsymbol{w}(0) = \boldsymbol{w}_0$
$E(\boldsymbol{w}(0)) = \textsc{OfflineError}(\mathscr{N}, \mathscr{V}, \boldsymbol{w}(0), P, d)$
while $t < T$ and $E(\boldsymbol{w}(t)) > \epsilon$ do
    compute weight update $\Delta\boldsymbol{w}(t)$ based on $E(\boldsymbol{w}(t))$
    $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta\boldsymbol{w}(t)$
    $E(\boldsymbol{w}(t+1)) = \textsc{OfflineError}(\mathscr{N}, \mathscr{V}, \boldsymbol{w}(t+1), P, d)$
    $t \leftarrow t+1$
end while
$\boldsymbol{w}_{opt} = \boldsymbol{w}(t)$

function $E(\boldsymbol{w}) = \textsc{OfflineError}(\mathscr{N}, \mathscr{V}, \boldsymbol{w}, P, d)$
    $E(\boldsymbol{w}) = 0$
    for $i = 1, \ldots, N_{tr}$ do
        evaluate output vector $\boldsymbol{y}_{\boldsymbol{p}_i}$, correspondent to input pattern $\boldsymbol{p}_i$
        $E(\boldsymbol{w}) \leftarrow E(\boldsymbol{w}) + d(\boldsymbol{y}_{\boldsymbol{p}_i}, \boldsymbol{t}_i)$
    end for
end function
```

weighted sum, so that the net input is given by (1.4), the second factor in (1.19) reads:

$$\frac{\partial u_{\boldsymbol{p},j}}{\partial w_{i,j}} = \frac{1}{\partial w_{i,j}} \left(\sum_{s \in S} w_{s,j}\, y_{\boldsymbol{p},s} - \theta_j \right) = y_{\boldsymbol{p},i}\,. \tag{1.20}$$

We then denote the opposite of the first term in (1.19) by $\delta_{\boldsymbol{p},j}$, i.e.,

$$\delta_{\boldsymbol{p},j} = -\frac{\partial E_{\boldsymbol{p}}(\boldsymbol{w})}{\partial u_{\boldsymbol{p},j}}\,. \tag{1.21}$$

$\delta_{\boldsymbol{p},j}$ is often referred to as the *sensitivity* of the specific error $E_{\boldsymbol{p}}$ with respect to neuron j. Plugging (1.21) and (1.20) into (1.18), then embedding the resulting equation into (1.17), the weight update can be cast in the form:

$$\Delta w_{i,j} = \eta \sum_{\boldsymbol{p} \in P} \delta_{\boldsymbol{p},j}\, y_{\boldsymbol{p},i}\,. \tag{1.22}$$

We now proceed to derive an operative formula for $\delta_{\boldsymbol{p},j}$. Consider the case j is an output neuron. Supposing a specific error of the form (1.15) and an identity output function, it follows:

$$\begin{aligned}\delta_{\boldsymbol{p},j} &= -\frac{\partial E_{\boldsymbol{p}}(\boldsymbol{w})}{\partial y_{\boldsymbol{p},j}} \, \frac{\partial y_{\boldsymbol{p},j}}{\partial u_{\boldsymbol{p},j}} \\ &= -\frac{1}{\partial y_{\boldsymbol{p},j}} \frac{1}{M_O} \sum_{k=1}^{M_O} \left(t_{\boldsymbol{p},k} - y_{\boldsymbol{p},k}\right)^2 \, \frac{\partial a_{\boldsymbol{p},j}}{\partial u_{\boldsymbol{p},j}} \\ &= \frac{2}{M_O} \left(t_{\boldsymbol{p},j} - y_{\boldsymbol{p},j}\right) \, \frac{\partial f_{act}(u_{\boldsymbol{p},j})}{\partial u_{\boldsymbol{p},j}} \\ &= \frac{2}{M_O} \left(t_{\boldsymbol{p},j} - y_{\boldsymbol{p},j}\right) \, f'_{act}(u_{\boldsymbol{p},j}) \,.\end{aligned} \tag{1.23}$$

It worths mention here that (1.23) implies the derivative f'_{act} of the activation function f_{act} with respect to its argument. This motivates the requirement of a differentiable transfer function.

On the other hand, Equation (1.23) does not apply when j lies within an hidden layer. In fact, no teaching input is provided for an hidden neuron. In that case, let us denote by $R = \{r_1, \dots, r_n\}$ the set of n neurons receveing the output generated by j, i.e., the *successors*. We then point out that the output of any neuron can directly affect only the neurons which receive the output itself, i.e., the successors [26]. Hence:

$$\begin{aligned}\delta_{\boldsymbol{p},j} &= \frac{\partial E_{\boldsymbol{p}}(\boldsymbol{w})}{\partial u_{\boldsymbol{p},j}} \\ &= \frac{\partial E_{\boldsymbol{p}}(u_{\boldsymbol{p},r_1}, \dots, u_{\boldsymbol{p},r_n})}{\partial u_{\boldsymbol{p},j}} \\ &= \frac{\partial E_{\boldsymbol{p}}(u_{\boldsymbol{p},r_1}, \dots, u_{\boldsymbol{p},r_n})}{\partial y_{\boldsymbol{p},j}} \, \frac{\partial y_{\boldsymbol{p},j}}{\partial u_{\boldsymbol{p},j}} \\ &= \sum_{k=1}^{n} \frac{\partial E_{\boldsymbol{p}}}{\partial u_{\boldsymbol{p},r_k}} \, \frac{\partial u_{\boldsymbol{p},r_k}}{\partial y_{\boldsymbol{p},j}} \, \frac{\partial y_{\boldsymbol{p},j}}{\partial u_{\boldsymbol{p},j}} \,.\end{aligned} \tag{1.24}$$

Applying the definition of sensitivity for neuron r_k, $k = 1, \dots, n$, under the same assumptions of Equation (1.23) we can further develop (1.24):

$$\begin{aligned}\delta_{\boldsymbol{p},j} &= \sum_{k=1}^{n} \left(-\delta_{\boldsymbol{p},r_k}\right) \, \frac{1}{\partial y_{\boldsymbol{p},j}} \left(\sum_{l=1}^{m_k} w_{s_l,r_k} y_{\boldsymbol{p},s_l} - \theta_{r_k}\right) \, \frac{\partial f_{act}(u_{\boldsymbol{p},j})}{\partial u_{\boldsymbol{p},j}} \\ &= -\sum_{k=1}^{n} \delta_{\boldsymbol{p},r_k} \, w_{j,r_k} \, f'_{act}(u_{\boldsymbol{p},j}) \,.\end{aligned} \tag{1.25}$$

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

Let us now collect and 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.,

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

Assume the accumulated mean squared error (1.16) as performance function. Understanding the dependence on time for the sake of clarity, the weight update $\Delta w_{i,j}$ reads:

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

where $\eta > 0$, while $\delta_{\boldsymbol{p},j}$ is given by

$$\delta_{\boldsymbol{p},j} = \begin{cases} f'_{act}(u_{\boldsymbol{p},j}) \sum\limits_{r \in R} \delta_{\boldsymbol{p},r} \; w_{j,r} \,, & \text{if } j \text{ inner neuron} \;\;, \qquad (1.26a) \\ \dfrac{2}{M_O} f'_{act}(u_{\boldsymbol{p},j}) \left(t_{\boldsymbol{p},j} - y_{\boldsymbol{p},j} \right) , & \text{if } j \text{ output neuron} \;\;. \qquad (1.26b) \end{cases}$$

Some relevant remarks about the overall algorithm should be pointed out. First, observe that Equation (1.26a) defines $\delta_{\boldsymbol{p},j}$ for a hidden node j by relying on neurons in the following layer, whereas Equation (1.26b) only involves variables concerning the neuron (namely, $u_{\boldsymbol{p},j}$ and $y_{\boldsymbol{p},j}$) and the exact output $t_{\boldsymbol{p},j}$. Therefore, the coupled equations (1.26a)-(1.26b) 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 [26, 45].

The weight updating procedure detailed above corresponds to the offline version of the backpropagation of error, since it involves the total error E. On the other hand, the online algorithm readly comes from Equation (1.22): simply drop the summation over the elements of the training set P.

Although intuitive and very promising, backpropagation of error suffers of all those 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 (1.17) of the learning rate $\eta \in (0, 1]$, acting as a reducing factor and thus enabling a keener control on the descent. As suggested by Kriesel [26], 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. 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 surface, thus to identify a region of interest. Then, the learning rate should be large, i.e., close to 1, thus to speed up the exploration. However, as we approach the optimal configuration, we may want to progressively reduce the learning rate, then the step size, to fine-tune the weights of the neural network.

Nevertheless, selecting an appropriate value for η is still more an art than a science. Furthermore, for sigmoidal activations functions as the ones shown in Figure **??**, the derivative is close to zero far from the origin. This results in the fact that it becomes very difficult to move neurons away from the limits of the activation, which could extremely extend the learning time [26]. Hence, different alternatives to backpropagation have been proposed in literature, either by modifying the original algorithm (as, e.g., the resilient backpropagation [40] or the quickpropagation [11]), or pursuing a different numerical approach to the optimization problem. The latter class of algorithms includes the Levenberg-Marquardt algorithm, which we shall extensively use in our numerical tests.

Levenberg-Marquardt algorithm

While backpropagation is a steepest descent algorithm, the Levenberg-Marquardt algorithm [28] is an approximation to the Newton's method [12]. As for backpropagation, the learning procedure is driven by the loss function $E = E(\boldsymbol{w})$, $\boldsymbol{w} \in |\mathcal{V}|$. Applying the Newton's method for the minimization of E, at each iteration the *search direction* $\Delta \boldsymbol{w}$ is found by solving the following linear system:

$$\nabla^2 E(\boldsymbol{w}) \Delta \boldsymbol{w} = -\nabla E(\boldsymbol{w}) , \tag{1.27}$$

where $\nabla E(\boldsymbol{w})$ and $\nabla^2 E(\boldsymbol{w})$ denotes, respectively, the gradient vector and the Hessian matrix of E with respect to its argument $\boldsymbol{w}$. Assume now that the loss function is represented as the accumulated mean squared error,

$$E(\boldsymbol{w}) = \sum_{\boldsymbol{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} \left(t_{\boldsymbol{p},j} - y_{\boldsymbol{p},j} \right)^2 = \sum_{\boldsymbol{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} e_{\boldsymbol{p},j}(\boldsymbol{w})^2 ; \tag{1.28}$$

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

$$J_{\boldsymbol{p}}(\boldsymbol{w}) = \begin{bmatrix} \dfrac{\partial e_{\boldsymbol{p},1}}{\partial w_1} & \dfrac{\partial e_{\boldsymbol{p},1}}{\partial w_2} & \cdots & \dfrac{\partial e_{\boldsymbol{p},1}}{\partial w_{|\mathcal{V}|}} \\ \dfrac{\partial e_{\boldsymbol{p},2}}{\partial w_1} & \dfrac{\partial e_{\boldsymbol{p},2}}{\partial w_2} & \cdots & \dfrac{\partial e_{\boldsymbol{p},2}}{\partial w_{|\mathcal{V}|}} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial e_{\boldsymbol{p},M_O}}{\partial w_1} & \dfrac{\partial e_{\boldsymbol{p},M_O}}{\partial w_2} & \cdots & \dfrac{\partial e_{\boldsymbol{p},M_O}}{\partial w_{|\mathcal{V}|}} \end{bmatrix} \in \mathbb{R}^{M_O \times |\mathcal{V}|} \tag{1.29}$$

simple computations yield:

$$\nabla E(\boldsymbol{w}) = \sum_{\boldsymbol{p} \in P} \frac{2}{M_0} J_{\boldsymbol{p}}(\boldsymbol{w})^T \boldsymbol{e_p} \in \mathbb{R}^{|\mathcal{V}|} \tag{1.30}$$

and

$$\nabla^2 E(\boldsymbol{w}) = \sum_{\boldsymbol{p} \in P} \frac{2}{M_O} \left[J_{\boldsymbol{p}}(\boldsymbol{w})^T J_{\boldsymbol{p}}(\boldsymbol{w}) + S(\boldsymbol{w}) \right] \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|} , \tag{1.31}$$

with

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

Assuming $S(\boldsymbol{w}) \approx 0$, inserting (1.30) and (1.31) into (1.27) we get the linear system

$$\left[\sum_{\boldsymbol{p} \in P} J_{\boldsymbol{p}}(\boldsymbol{w})^T J_{\boldsymbol{p}}(\boldsymbol{w}) \right] \Delta \boldsymbol{w} = - \sum_{\boldsymbol{p} \in P} J_{\boldsymbol{p}}(\boldsymbol{w})^T \boldsymbol{e_p}(\boldsymbol{w}) . \tag{1.32}$$

The Levenberg-Marquardt modification to the Newton's method reads [12, 28]:

$$\left[\sum_{\boldsymbol{p} \in P} J_{\boldsymbol{p}}(\boldsymbol{w})^T J_{\boldsymbol{p}}(\boldsymbol{w}) + \mu I \right] \Delta \boldsymbol{w} = - \sum_{\boldsymbol{p} \in P} J_{\boldsymbol{p}}(\boldsymbol{w})^T \boldsymbol{e_p}(\boldsymbol{w}) , \tag{1.33}$$

with $\mu \geq 0$ and I the identity matrix of size $|\mathcal{V}| \times |\mathcal{V}|$. Note that if $\mu = 0$ we recover the Newton's method (1.32), while for $\mu \gg 1$ the search direction $\Delta \boldsymbol{w}$ approaches the antigradient of E, i.e., we recover the backpropagation algorithm. Then, the Levenberg-Marquardt algorithm can be seen as an interpolation between the 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 (1.33). Whenever the step $\Delta\boldsymbol{w}(t)$ leads to a reduction in the performance function, i.e., $E(\boldsymbol{w}(t)+\Delta\boldsymbol{w}(t)) < E(\boldsymbol{w}(t))$, the parameter μ is reduced by a factor $\beta > 1$. Conversely, if $E(\boldsymbol{w}(t)+\Delta\boldsymbol{w}(t)) > E(\boldsymbol{w}(t))$ 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 the Newton's method, since the latter may diverge. Yet, once in a neighborhood of the minimum, we switch to the Newton's method so to exploit its faster convergence [28].

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

$$\frac{\partial e_{\boldsymbol{p},h}}{\partial w_k} = \frac{\partial e_{\boldsymbol{p},h}}{\partial w_{i,j}}, \quad 1 \le h \le M_O, 1 \le k \le |\mathcal{V}|. \tag{1.34}$$

We recognize that the derivative on the right-hand side of Equation (1.34) is intimately related with the gradient of the performance function. Therefore, we can follow the very same steps performed to derive the backpropagation, namely:

$$\begin{aligned}\frac{\partial e_{\boldsymbol{p},h}}{\partial w_{i,j}} &= \frac{\partial e_{\boldsymbol{p},h}}{\partial u_{\boldsymbol{p},j}} \frac{\partial u_{\boldsymbol{p},j}}{\partial w_{i,j}} \\ &= \frac{\partial e_{\boldsymbol{p},h}}{\partial y_{\boldsymbol{p},j}} \frac{\partial y_{\boldsymbol{p},j}}{\partial u_{\boldsymbol{p},j}} \frac{\partial u_{\boldsymbol{p},j}}{\partial w_{i,j}} \\ &= \frac{\partial e_{\boldsymbol{p},h}}{\partial y_{\boldsymbol{p},j}} f'_{act}(u_{\boldsymbol{p},j})\, y_{\boldsymbol{p},i} \\ &= \delta_{\boldsymbol{p},h,j}\, y_{\boldsymbol{p},i}\,,\end{aligned} \tag{1.35}$$

with

$$\delta_{\boldsymbol{p},h,j} := -\frac{\partial e_{\boldsymbol{p},h}}{\partial u_{\boldsymbol{p},j}} = -\frac{\partial e_{\boldsymbol{p},h}}{\partial y_{\boldsymbol{p},j}} f'_{act}(u_{\boldsymbol{p},j})\,. \tag{1.36}$$

For the computation of the derivative $\partial e_{\boldsymbol{p},h} \,/\, \partial y_{\boldsymbol{p},j}$ occurring in (1.36), further 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 $e_{\boldsymbol{p},h} = e_{\boldsymbol{p},j} = t_{\boldsymbol{p},j} - y_{\boldsymbol{p},j}$, then

$$\frac{\partial e_{\boldsymbol{p},h}}{\partial y_{\boldsymbol{p},j}} = -1\,; \tag{1.37}$$

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

$$\frac{\partial e_{\boldsymbol{p},h}}{\partial y_{\boldsymbol{p},j}} = 0\,; \tag{1.38}$$

(iii) j inner neuron: letting R be the set of successors of j, similarly to (1.25) the chain rule yields

$$\frac{\partial e_{\boldsymbol{p},h}}{\partial y_{\boldsymbol{p},j}} = \sum_{r \in R} \frac{\partial e_{\boldsymbol{p},h}}{\partial u_{\boldsymbol{p},r}} \frac{\partial u_{\boldsymbol{p},r}}{\partial y_{\boldsymbol{p},j}} = -\sum_{r \in R} \delta_{\boldsymbol{p},h,r} \; w_{j,r} \, . \tag{1.39}$$

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

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

with

$$\delta_{\boldsymbol{p},h,j} = \begin{cases} f'_{act}(u_{\boldsymbol{p},j}) \sum_{r \in R} \delta_{\boldsymbol{p},h,r} \; w_{j,r} \, , & \text{if } j \text{ inner neuron} \; , \quad (1.40a) \\ f'_{act}(u_{\boldsymbol{p},j}) \delta^K_{jh} \, , & \text{if } j \text{ output neuron} \; , \quad (1.40b) \end{cases}$$

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 **??**.

Let us finally remark that a trial and error approach is still required to find satisfactory values for μ and β; as proposed in [28], a good starting point may be $\mu = 0.01$, with $\beta = 10$. Moreover, the dimension of the system (1.33) increases nonlinearly with the number of neurons in the network, making the Levenberg-Marquardt algorithm poorly efficient for large networks [12]. However, it is more efficient than backpropagation for networks with a few hundreads of connections, besides producing much more accurate results. We shall gain further insights into this topic in Chapter **??**.

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

```
function $\left[\boldsymbol{w} + \Delta\boldsymbol{w}, \, E(\boldsymbol{w} + \Delta\boldsymbol{w}), \, \mu\right]$ = LMITERATION($\mathscr{N}, \, \mathscr{V}, \, \boldsymbol{w}, \, P, \, E(\boldsymbol{w}), \, d, \, \mu$)
    $\beta = 10$
    for $i = 1, \ldots, N_{tr}$ do
        evaluate output vector $\boldsymbol{y}_{\boldsymbol{p}_i}$, correspondent to input pattern $\boldsymbol{p}_i$
        $\boldsymbol{e}_{\boldsymbol{p}_i} = \boldsymbol{y}_{\boldsymbol{p}_i} - \boldsymbol{t}_{\boldsymbol{p}_i}$
        for $h = 1, \ldots, M_0$, $k = 1, \ldots, |\mathscr{V}|$ do
            compute $\left(J_{\boldsymbol{p}_i}\right)_{h,k}$ according to (1.35), (1.40a) and (1.40b)
        end for
    end for
    assemble and solve $\left[\sum_{i=1}^{N_{tr}} J^T_{\boldsymbol{p}_i} J_{\boldsymbol{p}_i} + \mu I\right] \Delta\boldsymbol{w} = -\sum_{i=1}^{N_{tr}} J^T_{\boldsymbol{p}_i} \boldsymbol{e}_{\boldsymbol{p}_i}$
    $E(\boldsymbol{w} + \Delta\boldsymbol{w})$ = OFFLINEERROR($\mathscr{N}, \, \mathscr{V}, \, \boldsymbol{w} + \Delta\boldsymbol{w}, \, P_{tr}, \, d$)
    if $E(\boldsymbol{w} + \Delta\boldsymbol{w}) < E(\boldsymbol{w})$ 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 multi-layer perceptrons. 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* [26]. 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. Then, too few neurons (i.e., synapses) do not endow the network with the necessary representation capability, leading to poor results, i.e., large values for the performance function. On the other hand, an oversized network would rather precisely align 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 decades, many expedients 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 $L(\boldsymbol{\mu})$, namely

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

Here, L is a functional increasing with the components of $\boldsymbol{w}$, thus penalizing large values for the weights, and so preventing the network from heavily relying on individual data points [29]. 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 [24]. 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 (1.33) yielding the weights and biases update $\Delta \boldsymbol{w}$. Whereas, 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 [29]. For further details on the way such subsets have been generated for our test cases, we refer the reader to Chapter **??**.

At the beginning of the learning stage, the error on both the training and validation data set typically decreases. However, while the error yielded by the training samples should keep lowering 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 keeps rising up for K_{ea} consecutive iterations. The optimal configuration $\boldsymbol{w}_{opt}$ is then that one yielding the minimum of the validation (and not the training) error curve.

Once in possession of an effective training procedure, one can basically 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 [18]) to iteratively remove the *less relevant* connection, erasing a neuron whenever it gets isolated from the rest of the network.

As we shall explain in Chapter **??**, in this work we resort to the first approach for the sake of implementation-wise 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 a 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 $\boldsymbol{w}_0$, we train each network topology several times, say K_{mr}, employing different $\boldsymbol{w}_0^k$, $k = 1, \dots, K_{mr}$, 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, based on the Levenberg-Marquardt algorithm and keeping into consideration all the observations made in this section, is given in Algorithm 1.4.

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

Algorithm 1.4 The complete training algorithm adopted in our numerical tests.

```
function $\left(\mathcal{N},\mathcal{V},\,\boldsymbol{w}_{opt}\right)$ = TRAINING($(\mathcal{N},\mathcal{V}),\ P_{tr},\ P_{va},\ P_{te},\ d,\ K_{ms},\ \epsilon,\ T,\ K_{ea}$)
    $E_{opt}=\infty$
    for $j=1,\dots,K_{ms}$ do
        $t=0,\ \ k=0,\ \ \mu=0.01$
        randomly generate $\boldsymbol{w}(0)$
        $E_{tr}(\boldsymbol{w}(0))$ = OFFLINEERROR($\mathcal{N},\mathcal{V},\,\boldsymbol{w}(0),\,P_{tr},\,d$)
        while $t<T$ and $E_{tr}(\boldsymbol{w}(t))>\epsilon$ and $k<K_{ea}$ do
            $\left[\boldsymbol{w}(t+1),\,E_{tr}(\boldsymbol{w}(t+1)),\,\mu\right]$ = LMITERATION($\mathcal{N},\mathcal{V},\,\boldsymbol{w}(t),\,P_{tr},\,E(\boldsymbol{w}(t)),\,d,\,\mu$)
            $E_{va}(\boldsymbol{w}(t+1))$ = OFFLINEERROR($\mathcal{N},\mathcal{V},\,\boldsymbol{w}(t+1),\,P_{va},\,d$)
            if $E_{va}(\boldsymbol{w}(t+1))>E_{va}(\boldsymbol{w}(t))$ then
                $k\leftarrow k+1$
            else
                $k=0$
            end if
            $t\leftarrow t+1$
        end while
        $\boldsymbol{w}_{opt}^{(j)}=\boldsymbol{w}(t-k)$
        $E_{te}\left(\boldsymbol{w}_{opt}^{(j)}\right)$ = OFFLINEERROR($\mathcal{N},\mathcal{V},\,\boldsymbol{w}_{opt}^{(j)},\,P_{te},\,d$)
        if $E_{te}\left(\boldsymbol{w}_{opt}^{(j)}\right)<E_{opt}$ then
            $\boldsymbol{w}_{opt}=\boldsymbol{w}_{opt}^{(j)}$
            $E_{opt}=E_{te}\left(\boldsymbol{w}_{opt}^{(j)}\right)$
        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) method with neural networks for the efficient resolution of parametrized nonlinear partial differential equations (PDEs) defined on variable shape domains. As illustrative yet relevant instances of nonlinear PDEs, the nonlinear Poisson equation and the stationary Navier-Stokes equations are used as expository vehicles. The latter features a quadratic nonlinearity laying in the convective term, while for the former the nonlinearity will stem from a solution-dependent viscosity (or diffusion coefficient). Throughout this work, we shall confine the attention to one- and two-dimensional differential problems. However, the proposed RB procedure can be readily extended to higher dimensions, and to other classes of nonlinear PDEs as well.

Before diving into the mathematical foundation of the proposed RB method, it worths provide an overview of the whole numerical procedure. In doing so, we seek to clarify the motivations for resorting to neural networks, and 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 $\mathscr{P}_g$ 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 then seek the corresponding solution $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$ of the underlying PDE in a suitable Hilbert space $\widetilde{V}(\boldsymbol{\mu}_g)$, defined over the computational domain $\widetilde{\Omega}(\boldsymbol{\mu}_g)$.

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

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

allowing to formulate and solve the differential problem over a fixed, parameter-independent, reference domain Ω [31], such that

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

Here, $u(\boldsymbol{\mu})$ represents the solution over the reference domain Ω, and lies in a Hilbert space

V. Then, $u(\boldsymbol{\mu})$ can be regarded as a map linking the parameter domain $\mathscr{P}$ with V, i.e.,

$$u : \mathscr{P} \to V \,. \tag{2.1}$$

The map (2.1) defines the *solution manifold* $\mathscr{M} = \{u = u(\boldsymbol{\mu}) \;:\; \boldsymbol{\mu} \in \mathscr{P}\} \subset V$, consisting of solutions to the parametrized PDE for any admissible input parameter [19]. 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 $\widetilde{\Omega}(\boldsymbol{\mu})$ [22].

In real life applications, differential equations often do not admit an analytical solution in closed form. Hence, one has typically to resort to some numerical techniques, e.g., the finite element (FE) method, providing a *high-fidelity* approximation $u_h(\boldsymbol{\mu})$ of $u(\boldsymbol{\mu})$. The discrete solution u_h is sought in a finite-dimensional space $V_h(\boldsymbol{\mu}_g) \subset V(\boldsymbol{\mu}_g)$, 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:

$$\boldsymbol{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^M \,. \tag{2.2}$$

For the FE method, $\mathbf{u}_h$ 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.

Suppose now we are concerned with the repeated resolution of the parametrized PDE for different parameter values. From a geometrical standpoint, we have then to span the *discrete solution manifold* $\mathscr{M}_h = \{u_h = u_h(\boldsymbol{\mu}) \;:\; \boldsymbol{\mu} \in \mathscr{P}\} \subset V_h$. To this end, at the algebraic level this would imply the resolution of a possibly large number of nonlinear systems as (2.2). 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 [38].

In this scenario, *reduced order modeling* (ROM) (also known as *order model reduction*) aims at replacing the computationally expensive discrete model, encoded by the large-scale system (2.2), with a reduced problem of significant smaller dimension L, with *L independent of M*. The variegate ROM methods proposed in 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 $\mathscr{M}_h$ and generated by L *basis functions* $\{\psi_1, \dots, \psi_L\} \subset V_h$, namely

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

Then, 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

$$\mathbf{u}_L = \mathbb{V}\, \mathbf{u}_{\mathtt{rb}} = \sum_{i=1}^{L} u_{\mathtt{rb}}^{(i)} \boldsymbol{\psi}_i \,, \tag{2.3}$$

so that

$$u_L(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{i=1}^{L} u_{\mathtt{rb}}^{(i)}(\boldsymbol{\mu}) \psi_i(\boldsymbol{x}) \,. \tag{2.4}$$

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

$$\boldsymbol{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L. \tag{2.5}$$

The reduced system (2.5) derives from the large-scale system (2.2) by replacing $\mathbf{u}_h$ with $\mathbf{u}_L$ and enforcing, for each equation, the orthogonality of the residual to the column space $\mathrm{Col}(\mathbb{V})$ of $\mathbb{V}$[1] [31]. 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 [36]. 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 (2.2). In this thesis, we refer to the well-known Proper Orthogonal Decomposition (POD) algorithm for the generation of the reduced basis, see Section 2.5.1. However, independently of the specific method in use, it worths notice that any reduced order strategy should cooperate with the underpinning full-order model to generate reliable results [19]. On the other hand, 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 the system (2.5). 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 its cost is uniquely related to the dimension L of the reduced space $V_{\mathtt{rb}}$ [38]. Whereas, to accomplish this complete decoupling in a nonlinear, non-affine case, one has to recover an affine approximation of the differential operator [31]. In this respect, the discrete empirical interpolation method (DEIM) (see, e.g., [2]) and its matrix version (MDEIM) (see, e.g., [34]) represent the state-of-the-art nowadays.

In the last decade, the development of a priori and a posteriori error estimators for RB methods have received an increasing attention. Indeed, such estimators, bounding the error between the reduced solution u_L and the underlying continuous 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 [19]. Nevertheless, estimates available in literature mainly address linear affine problems, then simpler than those tackled in this work. Only recently there have been attempts to extend existing convergence theories to nonlinear, non-affine problems, e.g., the Navier-Stokes equations [38], relying on advanced functional tools. In any case, error estimators do not account for possible numerical instabilities introduced by the nonlinear solver, e.g., the Newton's method, which is applied to the reduced system (2.5). In other terms, it is implicitly assumed that the reduced system, either linear or nonlinear, is solved *exactly*. However, iterative methods may fail to converge to the exact solution of the reduced system. As we shall further investigate in Section 2.6, this could be ascribed to the dense nature of the Jacobian of the system. Even before, we should analyze the way the reduced system is constructed: projecting the full-order system (2.2)

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

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

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

onto Col($\mathbb{V}$) does not ensure that the exact solution to (2.5) is represented by the projection onto Col($\mathbb{V}$) of the high-fidelity solution $\mathbf{u}_h$, i.e., the following may *a priori* hold:

$$\mathbf{u}_L(\boldsymbol{\mu}) = \mathbb{V}\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) \neq \mathbb{V}\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}) \quad . \tag{2.6}$$

Here, $\mathbb{P} = \mathbb{V}\mathbb{V}^T \,:\, \mathbb{R}^M \to \mathrm{Col}(\mathbb{V})$ is the projection operator onto Col($\mathbb{V}$). Therefore, $\mathbb{P}\mathbf{u}_h$ represents the element of Col($\mathbb{V}$) which best approximates $\mathbf{u}_h$ (in the Euclidean norm).

This motivates the research for alternative methods to compute a reduced basis solution in the online stage, and which could overcome the limits of the algebraic way. For this purpose, let us observe that (2.6) can be equivalently stated as:

$$\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}) \neq \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}), \tag{2.7}$$

where $\mathbb{V}^T\mathbf{u}_h$ are the coefficients for the expansion of $\mathbb{P}\mathbf{u}_h$ in terms of the reduced basis vectors $\{\boldsymbol{\psi}_1, \dots, \boldsymbol{\psi}_L\}$, i.e.,

$$\mathbb{P}\mathbf{u}_h(\boldsymbol{\mu}) = \sum_{i=1}^{L} [\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})]_i \boldsymbol{\psi}_i \,. \tag{2.8}$$

The key idea is then to seek an approximation $\hat{\boldsymbol{\pi}}$ to the map $\boldsymbol{\pi}$, defined as

$$\begin{aligned} \boldsymbol{\pi} \,:\, \mathscr{P} \subset \mathbb{R}^P &\to \mathbb{R}^L \\ \boldsymbol{\mu} &\mapsto \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}) \,. \end{aligned} \tag{2.9}$$

The approximation might be accomplished either through *interpolation* [1, 6] or *regression* [14] of a collection of target input-output pairs $\{\boldsymbol{\mu}^{(i)}, \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^{N_{tr}}$. In this work, we pursue the latter approach, employing multi-layer feedforward neural networks, presented in Chapter 1. In Section 2.6, we shall 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 discussed as well.

We now proceed with a deeper theoretical presentation of the elements mentioned so far. First, we rigorously define the functional framework for the subsequent analyses. Let us observe that throughout the chapter we seek to develop a theory which is as comprehensive as possible, and which includes as specific cases the two classes of PDEs considered in this project (i.e., the nonlinear Poisson equation and the steady Navier-Stokes equations). In doing so, we aim at proving how the proposed methodology could be almost effortlessy extended to other differential problems.

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 $\widetilde{\Omega}(\boldsymbol{\mu}_g)$, and 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 $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P}$, seek $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{G}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) = 0 \quad \text{in } \widetilde{V}'(\boldsymbol{\mu}_g)\,, \tag{2.10}$$

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 the problem (2.10) to be stated in a weak (or variational) form [37]. 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 (2.10) then reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathscr{P}$, seek $\widetilde{u}(\boldsymbol{\mu}) \in \widetilde{V}(\boldsymbol{\mu}_g)$ such that

$$\widetilde{g}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = 0 \quad \forall v \in \widetilde{V}\,. \tag{2.11}$$

Note that the definition of $\widetilde{g}$ 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 (2.10) and (2.11), in this work we focus on two relevant examples - the nonlinear Poisson equation and the stationary Navier-Stokes equations - which will serve as convenient test cases in Chapter **??**.

2.1.1 Nonlinear Poisson equation

Despite a rather simple form, the Poisson equation has proved effective for the modelization of steady phenomena occurring in many application fields, as, e.g., electromagnetism, heat transfer, and underground flows [30]. Throughout the text, we consider the following version of the parametrized Poisson equation for a state variable $\widetilde{u} = \widetilde{u}(\boldsymbol{\mu})$:

$$\begin{cases} -\widetilde{\nabla} \cdot \left(\widetilde{k}(\widetilde{\boldsymbol{x}}, \widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \right) = \widetilde{s}(\widetilde{\boldsymbol{x}}; \boldsymbol{\mu}_{ph}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g)\,, & (2.12a) \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{g}(\widetilde{\boldsymbol{\sigma}}; \boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}_D\,, & (2.12b) \\ \widetilde{k}(\widetilde{\boldsymbol{\sigma}}, \widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} \widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\boldsymbol{n}} = 0 & \text{on } \widetilde{\Gamma}_N\,. & (2.12c) \end{cases}$$

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 viscosity, $\widetilde{s} : \widetilde{\Omega} \times \mathscr{P}_{ph} \to \mathbb{R}$ 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 $\boldsymbol{\mu} \in \mathscr{P}$, then set

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

i.e., the space of squared integrable functions, together with their first (distributional) derivatives, which vanish on $\widetilde{\Gamma}_D$. Multiplying (2.12a) by a *test* function $v \in \widetilde{V}$, integrating over $\widetilde{\Omega}$, and exploiting integration by parts on the left-hand side, yields:

$$\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(\widetilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) = \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph})\; v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g), \tag{2.13}$$

where we have omitted the dependence on the space variable $\widetilde{\boldsymbol{x}}$ for ease of notation. For the integrals in Equation (2.13) to be well-defined, we may 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 $u_{\widetilde{g}} = u_{\widetilde{g}}(\boldsymbol{\mu}) \in H^1\big(\widetilde{\Omega}(\boldsymbol{\mu}_g)\big)$ be a *lifting* function such that $u_{\widetilde{g}}(\boldsymbol{\mu})\big|_{\widetilde{\Gamma}_D} = \widetilde{g}(\boldsymbol{\mu}_{ph})$, with $\widetilde{g}(\boldsymbol{\mu}_{ph}) \in H^{1/2}\big(\widetilde{\Gamma}_N\big)^3$. We assume that such a function can be effortlessy construct, e.g., by interpolation of the boundary condition. Hence, upon defining

$$\begin{aligned}
\widetilde{a}(w, v; \boldsymbol{\mu}) := &\int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + u_{\widetilde{g}}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} u_{\widetilde{g}}(\boldsymbol{\mu}) \cdot \widetilde{\nabla} v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) \\
&+ \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{k}(w + u_{\widetilde{g}}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph})\, \widetilde{\nabla} w \cdot \widetilde{\nabla} v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall\, w, v \in \widetilde{V}(\boldsymbol{\mu}_g),
\end{aligned} \tag{2.14a}$$

$$\widetilde{f}(v; \boldsymbol{\mu}) := \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{s}(\boldsymbol{\mu}_{ph})\; v \; d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall\, v \in \widetilde{V}(\boldsymbol{\mu}_g), \tag{2.14b}$$

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

$$\widetilde{a}(\widetilde{u}(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = \widetilde{f}(v; \boldsymbol{\mu}) \quad \forall\, v \in \widetilde{V}(\boldsymbol{\mu}_g), \tag{2.15}$$

Then, the weak solution of problem (2.12) is given by $\widetilde{u}(\boldsymbol{\mu}) + u_{\widetilde{g}}(\boldsymbol{\mu})$. Note that resorting to a lifting function makes the formulation (2.15) *symmetric*, i.e., both the solution and the test functions are picked up from the same functional space.

Lastly, let us remark that the weak formulation (2.15) can be cast in the form (2.11) 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 $H^{1/2}\big(\widetilde{\Gamma}_N\big)$:

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

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 $\widetilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 2, 3$ [39]. 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 shall consider for our numerical tests reads:

$$
\begin{cases}
\widetilde{\nabla} \cdot \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = 0 & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g)\,, & (2.16a) \\
-\nu(\boldsymbol{\mu})\, \widetilde{\Delta} \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + (\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \widetilde{\nabla})\, \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \dfrac{1}{\rho(\boldsymbol{\mu})} \widetilde{\nabla} \widetilde{p}(\boldsymbol{\mu}) = \boldsymbol{0} & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}_g)\,, & (2.16b) \\
\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) = \widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph}) & \text{on } \widetilde{\Gamma}(\boldsymbol{\mu}_g)\,, & (2.16c) \\
\widetilde{p}(\boldsymbol{\mu})\widetilde{\boldsymbol{n}} - \nu(\boldsymbol{\mu}) \widetilde{\nabla} \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) \cdot \widetilde{\boldsymbol{n}} = \boldsymbol{0} & \text{on } \widetilde{\Gamma}_N(\boldsymbol{\mu}_g)\,. & (2.16d)
\end{cases}
$$

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 $\nu(\boldsymbol{\mu})$ represents the density and the kinematic viscosity of the fluid, respectively, and we suppose they are uniform over the domain. Note that, despite they encode physical properties, we let $\rho(\boldsymbol{\mu})$ and $\nu(\boldsymbol{\mu})$ 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 (2.16) with respect to $\boldsymbol{\mu}_g$ may be carried out by adapting either $\rho(\boldsymbol{\mu})$ or $\nu(\boldsymbol{\mu})$ as $\boldsymbol{\mu}_g$ varies, so to preserve a dimensionless quantity of interest; we refer the reader to Section **??** for a practical example.
For our purposes, it worths notice that the nonlinearity of problem (2.16) lies in the convective term

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

which shows up a *quadratic* nonlinearity. Conversely, both the first and third term of the momentum equation (2.16b) are linear in the solution $(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{p}(\boldsymbol{\mu}))$, as Equation (2.16a), which enforces mass conservation [38].

Let us now introduce the velocity space $\widetilde{X}(\boldsymbol{\mu}_g) = \left[H^1_{\widetilde{\Gamma}}\big(\widetilde{\Omega}(\boldsymbol{\mu}_g)\big)\right]^d$ 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 (2.16) is given by: for a fixed $\boldsymbol{\mu} \in \mathscr{P}$, seek $(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{p}(\boldsymbol{\mu})) \in \widetilde{X}(\boldsymbol{\mu}_g) \times \widetilde{Q}(\boldsymbol{\mu}_g)$ such that

$$
\widetilde{d}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{c}(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \widetilde{b}(\widetilde{p}(\boldsymbol{\mu}), \widetilde{\nabla} \cdot \boldsymbol{\chi}; \boldsymbol{\mu}) = \widetilde{f}_1(\boldsymbol{\chi}; \boldsymbol{\mu}) \qquad \forall \boldsymbol{\chi} \in \widetilde{X}(\boldsymbol{\mu}_g)\,, \qquad (2.17a)
$$

$$
\widetilde{b}(\widetilde{\nabla} \cdot \widetilde{\boldsymbol{v}}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) = \widetilde{f}_2(\xi; \boldsymbol{\mu}) \qquad \forall \xi \in \widetilde{Q}(\boldsymbol{\mu})\,, \qquad (2.17b)
$$

with the trilinear form $\widetilde{c}(\cdot, \cdot, \cdot; \boldsymbol{\mu})$, the bilinear forms $\widetilde{d}(\cdot, \cdot; \boldsymbol{\mu})$ and $\widetilde{b}(\cdot, \cdot; \boldsymbol{\mu})$, and the functionals $\widetilde{f}_1(\cdot; \boldsymbol{\mu})$ and $\widetilde{f}_2(\cdot; \boldsymbol{\mu})$ defined as follows:

$$
\begin{aligned}
\widetilde{c}(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = & \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \big(\boldsymbol{\psi} \cdot \widetilde{\nabla}\big)\, \boldsymbol{\chi} \cdot \boldsymbol{\eta}\, d\widetilde{\Omega}(\boldsymbol{\mu}_g) + \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \big(\boldsymbol{v}_{\widetilde{\boldsymbol{g}}} \cdot \widetilde{\nabla}\big)\, \boldsymbol{\chi} \cdot \boldsymbol{\eta}\, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \\
& + \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \big(\boldsymbol{\psi} \cdot \widetilde{\nabla}\big)\, \boldsymbol{v}_{\widetilde{\boldsymbol{g}}} \cdot \boldsymbol{\eta}\, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in \widetilde{X}(\boldsymbol{\mu}_g)\,, \qquad (2.18a)
\end{aligned}
$$

$$\widetilde{d}(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \nu(\boldsymbol{\mu}) \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \widetilde{\nabla}\boldsymbol{\psi} : \widetilde{\nabla}\boldsymbol{\chi} \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in \widetilde{X}(\boldsymbol{\mu}_g), \tag{2.18b}$$

$$\widetilde{b}(\boldsymbol{\psi}, \xi; \boldsymbol{\mu}) = -\frac{1}{\rho(\boldsymbol{\mu})} \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \left(\widetilde{\nabla} \cdot \boldsymbol{\psi}\right) \xi \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi} \in \widetilde{X}(\boldsymbol{\mu}_g), \forall \xi \in \widetilde{Q}(\boldsymbol{\mu}_g), \tag{2.18c}$$

$$\widetilde{f}_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = \widetilde{d}(\boldsymbol{v}_{\widetilde{\boldsymbol{g}}}, \boldsymbol{\psi}; \boldsymbol{\mu}) - \int_{\widetilde{\Omega}(\boldsymbol{\mu}_g)} \left(\boldsymbol{v}_{\widetilde{\boldsymbol{g}}} \cdot \widetilde{\nabla}\right) \boldsymbol{v}_{\widetilde{\boldsymbol{g}}} \cdot \boldsymbol{\psi} \, d\widetilde{\Omega}(\boldsymbol{\mu}_g) \qquad \forall \boldsymbol{\psi} \in \widetilde{X}(\boldsymbol{\mu}_g), \tag{2.18d}$$

$$\widetilde{f}_2(\xi; \boldsymbol{\mu}) = -\widetilde{b}(\boldsymbol{v}_{\widetilde{\boldsymbol{g}}}, \xi; \boldsymbol{\mu}) \qquad \forall \xi \in \widetilde{Q}(\boldsymbol{\mu}_g). \tag{2.18e}$$

(Recall that the *double dot* product $A : B$ between two square matrices $A, B \in \mathbb{R}^{n \times n}$ is defined as $A : B := \sum_{i,j=1}^{n} A_{ij} \, B_{ij}$). The function $\boldsymbol{v}_{\widetilde{\boldsymbol{g}}} = \boldsymbol{v}_{\widetilde{\boldsymbol{g}}}(\boldsymbol{\mu}_{ph}) \in \left[H^1\left(\widetilde{\Omega}(\boldsymbol{\mu}_g)\right)\right]^d$ appearing in Equations (2.18a), (2.18d) and (2.18e) acts as a lifting function, i.e. $\boldsymbol{v}_{\widetilde{\boldsymbol{g}}}|_{\widetilde{\Gamma}_D} = \widetilde{\boldsymbol{g}}(\boldsymbol{\mu}_{ph})$, so that the weak solution for the boundary value problem (2.16) is obtained as $(\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \boldsymbol{v}_{\widetilde{\boldsymbol{g}}}(\boldsymbol{\mu}_{ph}), \widetilde{p}(\boldsymbol{\mu}))$. In order to derive the weak formulation (2.17), one may proceed as done for the Poisson equation. First multiply (2.17a) and (2.17b) by trial functions $\boldsymbol{\chi} \in \widetilde{X}(\boldsymbol{\mu}_g)$ and $\xi \in \widetilde{Q}(\boldsymbol{\mu}_g)$, respectively, then integrate over $\widetilde{\Omega}(\boldsymbol{\mu}_g)$ and exploit integration by parts for the term

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

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 abusion of notation) replace $\widetilde{\boldsymbol{v}}(\boldsymbol{\mu})$ with $\widetilde{\boldsymbol{v}}(\boldsymbol{\mu}) + \boldsymbol{v}_{\widetilde{\boldsymbol{g}}}(\boldsymbol{\mu}_{ph})$ and plug the homogeneous Neumann boundary conditions into the integral equations.

We conclude this section showing how the variational problem (2.17) can be recast into the general form (2.11). 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

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

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 the differential problem at hand 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, expensive 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 $\left\{\widetilde{u}_h\left(\boldsymbol{\mu}^{(1)}\right), \dots, \widetilde{u}_h\left(\boldsymbol{\mu}^{(N)}\right)\right\}$, called *snapshots*, respectively correspondent to the parameter values $\left\{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\right\}$. However, recall that we are concerned with boundary value problems for stationary PDEs defined on variable shape geometries, so that any two snapshots $\widetilde{u}_h\big(\boldsymbol{\mu}^{(j)}\big)$ and $\widetilde{u}_h\big(\boldsymbol{\mu}^{(k)}\big)$, with $1 \leq j, k \leq N$ and $j \neq k$, are likely to have been computed on two different domains $\widetilde{\Omega}\big(\boldsymbol{\mu}_g^{(j)}\big)$ and $\widetilde{\Omega}\big(\boldsymbol{\mu}_g^{(k)}\big)$, respectively. Hence, the underlying high-fidelity solver should be carefully design so to guarantee the *compatibility* among snapshots. In particular, let $\mathbf{u}_h\big(\boldsymbol{\mu}^{(j)}\big)$ and $\mathbf{u}_h\big(\boldsymbol{\mu}^{(k)}\big)$ be the vectors collecting the degrees of freedom for $\widetilde{u}_h\big(\boldsymbol{\mu}^{(j)}\big)$ and $\widetilde{u}_h\big(\boldsymbol{\mu}^{(j)}\big)$, respectively. Then, we should ensure that:

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

(b) correspondent entries of the two vectors must be correlated in some sense.

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 accross 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 $\boldsymbol{\Phi} : \Omega \times \mathscr{P}_g \rightarrow \widetilde{\Omega}$ such that

$$\widetilde{\Omega}(\boldsymbol{\mu}_g) = \boldsymbol{\Phi}(\Omega; \boldsymbol{\mu}_g) . \tag{2.19}$$

As we shall prove in Sections 2.2.1 and 2.2.2, the transformation $\boldsymbol{\Phi}$ allows to restate the general problem (2.10) as follows. Let V be a suitable Hilbert space over Ω and V' its dual. Suppose V equipped with the scalar product $(\cdot,\cdot)_V : V \times V \rightarrow \mathbb{R}$ and the induced norm $\|\cdot\|_V = \sqrt{(\cdot,\cdot)_V} : V \rightarrow [0,\infty)$. Given the parametrized map $G : V \times \mathscr{P} \rightarrow V'$ 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

$$G(u(\boldsymbol{\mu}); \boldsymbol{\mu}) = 0 \quad \text{in } V' . \tag{2.20}$$

The weak formulation of problem (2.20) reads: given $\boldsymbol{\mu} \in \mathscr{P}$, seek $u(\boldsymbol{\mu}) \in V$ such that

$$g(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = 0 \quad \forall v \in V , \tag{2.21}$$

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

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

with $\langle\cdot,\cdot\rangle_{V',V} : V' \times V \rightarrow \mathbb{R}$ the dual pairing between V and V'. Let us point out that the explicit expression of the form $g(\cdot,\cdot; \boldsymbol{\mu})$ entails the map $\boldsymbol{\Phi}$, thus keeping trace 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

$$\widetilde{u}(\boldsymbol{\mu}) = u(\boldsymbol{\mu}) \circ \boldsymbol{\Phi}(\boldsymbol{\mu}) . \tag{2.22}$$

Upon resorting to the parametrized map $\boldsymbol{\Phi}$, for any $\boldsymbol{\mu} \in \mathscr{P}$ a discrete solution $u_h(\boldsymbol{\mu})$ to the problem (2.20) is sought on a *parameter-independent* cover Ω_h of the domain Ω. Provided a convenient choice for Ω, this eases the mesh generation process. In addition, we note that discretizing the problem (2.20) over Ω_h is equivalent to approximate the original problem (2.10) over the mesh $\widetilde{\Omega}_h(\boldsymbol{\mu}_g)$, given by

$$\widetilde{\Omega}_h(\boldsymbol{\mu}_g) = \boldsymbol{\Phi}(\Omega_h; \boldsymbol{\mu}_g) . \tag{2.23}$$

Therefore, the requirements (a) and (b) are automatically fulfilled provided that 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 [22] and whose construction is detailed in Section 2.2.3. Whereas, in the following section we introduce the mathematical tools required to re-state over the fixed domain the weak formulations (2.15) and (2.17) 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 of them. 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.

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

As a side remark, note that we understand the dependence of $\widetilde{\Omega}$ and $\boldsymbol{\Phi}$ on the parameter $\boldsymbol{\mu}_g$, since redundant for the following discussion.

Dealing with differential operators, we first need to relate the gradients with respect to $\widetilde{\boldsymbol{x}}$ and $\boldsymbol{x}$. To this end, let $\widetilde{\psi} : \widetilde{\Omega} \to \mathbb{R}$ a differentiable function. Then:

$$\frac{\partial \widetilde{\psi}(\widetilde{\boldsymbol{x}})}{\partial \widetilde{x}_i} = \frac{\partial \psi(\boldsymbol{\Phi}^{-1}(\boldsymbol{x}))}{\partial \widetilde{x}_i} = \sum_{j=1}^{d} \frac{\partial \psi(\boldsymbol{x})}{\partial x_j} \, \frac{\partial x_j}{\partial \widetilde{x}_i}, \quad 1 \leq i \leq d. \tag{2.24}$$

Introducing the Nabla 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

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

the compact form of (2.24) reads:

$$\widetilde{\nabla} \widetilde{\psi}(\widetilde{\boldsymbol{x}}) = \mathbb{J}_{\boldsymbol{\Phi}^{-1}}^T(\widetilde{\boldsymbol{x}}) \nabla \psi(\boldsymbol{x}). \tag{2.25}$$

Thanks to the invertibility of $\boldsymbol{\Phi}$, the inverse function theorem [42] ensures that

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

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

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

Then, Equation (2.25) gets:

$$\widetilde{\nabla}\widetilde{\psi}(\widetilde{\boldsymbol{x}}) = \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\widetilde{\boldsymbol{x}})\nabla\psi(\boldsymbol{x}). \tag{2.26}$$

Another major ingredient we need is the change of variables formula for integrals defined over $\widetilde{\Omega}$. Indeed, both formulations (2.15) and (2.17) involves integral forms. Letting $\widetilde{\psi}$ be a continuous and integrable function over $\widetilde{\Omega}$, we have [42]:

$$\int_{\widetilde{\Omega}} \widetilde{\psi}(\widetilde{\boldsymbol{x}})\, d\widetilde{\Omega} = \int_{\Omega} \psi(\boldsymbol{x}) |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{x})|\, d\Omega. \tag{2.27}$$

Here, $|\mathbb{J}_{\boldsymbol{\Phi}}|$ 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 (2.26) and (2.27) still holds when, e.g., $\psi \in H^1(\Omega)$ and $\widetilde{\psi} \in H^1(\widetilde{\Omega})$[4], provided that the equality in (2.26) is understood in the $L^2(\Omega)$ sense, and that the integrals in (2.27) are Lebesgue integrals. Actually, we can further relaxing the assumptions for the change of variables (2.27), just 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:

$$\int_{\widetilde{\Omega}} \widetilde{\boldsymbol{\psi}}\, d\widetilde{\Omega} = \int_{\Omega} \boldsymbol{\psi} |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega. \tag{2.28}$$

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

Upon these considerations, the equalities hereunder follows from a straightforward combination of (2.26) with (2.27) and (2.28):

$$\int_{\widetilde{\Omega}} \widetilde{\nabla}\widetilde{\psi} \cdot \widetilde{\nabla}\widetilde{\chi}\, d\widetilde{\Omega} = \int_{\Omega} \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\psi \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\chi |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\psi}, \widetilde{\chi} \in H^1(\widetilde{\Omega}), \tag{2.29}$$

$$\int_{\widetilde{\Omega}} \left(\widetilde{\boldsymbol{\psi}} \cdot \widetilde{\nabla}\right) \widetilde{\boldsymbol{\chi}} \cdot \widetilde{\boldsymbol{\eta}}\, d\widetilde{\Omega} = \int_{\Omega} \left(\boldsymbol{\psi} \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\right) \boldsymbol{\chi} \cdot \boldsymbol{\eta} |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\boldsymbol{\psi}}, \widetilde{\boldsymbol{\chi}}, \widetilde{\boldsymbol{\eta}} \in \left[H^1(\widetilde{\Omega})\right]^d, \tag{2.30}$$

$$\int_{\widetilde{\Omega}} \widetilde{\nabla}\widetilde{\boldsymbol{\psi}} : \widetilde{\nabla}\widetilde{\boldsymbol{\chi}}\, d\widetilde{\Omega} = \int_{\Omega} \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\boldsymbol{\psi} : \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\boldsymbol{\chi} |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\boldsymbol{\psi}}, \widetilde{\boldsymbol{\chi}} \in \left[H^1(\widetilde{\Omega})\right]^d, \tag{2.31}$$

$$\int_{\widetilde{\Omega}} \left(\widetilde{\nabla} \cdot \widetilde{\boldsymbol{\psi}}\right) \widetilde{\chi}\, d\widetilde{\Omega} = \int_{\Omega} \left(\mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla \cdot \boldsymbol{\psi}\right) \chi |\mathbb{J}_{\boldsymbol{\Phi}}|\, d\Omega \qquad \forall \widetilde{\boldsymbol{\psi}} \in \left[H^1(\widetilde{\Omega})\right]^d, \forall \widetilde{\chi} \in H^1(\widetilde{\Omega}). \tag{2.32}$$

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

2.2.2 The problems of interest

Exploiting the results derived in the previous section, let us now cast the weak formulations (2.15) and (2.17) for the Poisson equation and the steady Navier-Stokes equations, respectively, into the general variational problem (2.21). For this purpose, let Γ_D and Γ_N be the portions of the boundary $\Gamma = \partial\Omega$ on which we impose Dirichlet and Neumann boundary conditions, respectively. Setting $V = H^1_{\Gamma_D}(\Omega)$ and combining (2.14), (2.27) and (2.29), the variational formulation of the Poisson problem (2.12) over the reference domain Ω reads: given $\boldsymbol{\mu} \in \mathscr{P}$, seek $u(\boldsymbol{\mu}) \in V$ such that

$$a(u(\boldsymbol{\mu}), v; \boldsymbol{\mu}) = f(v; \boldsymbol{\mu}) \quad \forall v \in V, \tag{2.33}$$

with

$$\begin{aligned} a(w, v; \boldsymbol{\mu}) = & \int_\Omega k(w + u_g(\boldsymbol{\mu}); \boldsymbol{\mu})\; \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla w \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla v\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\,|\, d\Omega \\ & + \int_\Omega k(w + u_g(\boldsymbol{\mu}); \boldsymbol{\mu})\; \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla u_g(\boldsymbol{\mu}) \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla v\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})\,|\, d\Omega, \end{aligned} \tag{2.34a}$$

$$f(v; \boldsymbol{\mu}) = \int_\Omega s(\boldsymbol{\mu})\; v\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega, \tag{2.34b}$$

for any $w,\, v \in V$ and $\boldsymbol{\mu} \in \mathscr{P}$. Let us remark that, once on a parameter-independent configuration, we can avoid to distinguish between physical and geometrical parameters, since even the latter now affect the integrands, and not the domain of integration, in Equations (2.34a) and (2.34b). 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 (2.34) we resort to a lifting function $u_g(\boldsymbol{\mu}) \in H^1(\Omega)$ with $u_g(\boldsymbol{\mu})\big|_{\Gamma_D} = g(\boldsymbol{\mu})$ such that the weak solution to the problem (2.12) re-stated over Ω is obtained as $u(\boldsymbol{\mu}) + u_g(\boldsymbol{\mu})$.

Similarly, the variational formulation over Ω of the boundary value problem (2.16) for the stationary incompressible Navier-Stokes equations follow from (2.17), (2.36), (2.30), (2.31) and (2.32). Consider the velocity space $X = \big[H^1_{\Gamma_D}(\Omega)\big]^d$ 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

$$d(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + b(p(\boldsymbol{\mu}), \nabla \cdot \boldsymbol{\chi}; \boldsymbol{\mu}) = f_1(\boldsymbol{\chi}; \boldsymbol{\mu}) \qquad \forall \boldsymbol{\chi} \in X, \tag{2.35a}$$

$$b(\nabla \cdot \boldsymbol{v}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) = f_2(\xi; \boldsymbol{\mu}) \qquad \forall \xi \in Q, \tag{2.35b}$$

with, for any $\boldsymbol{\mu} \in \mathscr{P}$,

$$\begin{aligned} c(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = & \int_\Omega \big(\boldsymbol{\psi} \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\big)\, \boldsymbol{\chi} \cdot \boldsymbol{\eta}\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega + \int_\Omega \big(\boldsymbol{v_g} \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\big)\, \boldsymbol{\chi} \cdot \boldsymbol{\eta}\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \\ & + \int_\Omega \big(\boldsymbol{\psi} \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\big)\, \boldsymbol{v_g} \cdot \boldsymbol{\eta}\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in X, \end{aligned} \tag{2.36a}$$

$$d(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \nu(\boldsymbol{\mu}) \int_\Omega \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\boldsymbol{\psi} \,:\, \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\boldsymbol{\chi}\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \qquad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in X, \tag{2.36b}$$

$$b(\boldsymbol{\psi}, \xi; \boldsymbol{\mu}) = -\frac{1}{\rho(\boldsymbol{\mu})} \int_\Omega \big(\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla \cdot \boldsymbol{\psi}\big)\; \xi\, ,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\, d\Omega \qquad \forall \boldsymbol{\psi} \in X, \forall \xi \in Q, \tag{2.36c}$$

$$f_1(\boldsymbol{\psi};\boldsymbol{\mu}) = d(\boldsymbol{v_g},\boldsymbol{\psi};\boldsymbol{\mu}) - \int_\Omega \left(\boldsymbol{v_g}\cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\right)\boldsymbol{v_g}\cdot\boldsymbol{\psi}\,|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,d\Omega \qquad \forall\boldsymbol{\psi}\in X, \quad (2.36\text{d})$$

$$f_2(\xi;\boldsymbol{\mu}) = -b(\boldsymbol{v_g},\xi;\boldsymbol{\mu}) \qquad \forall\xi\in Q. \quad (2.36\text{e})$$

As usual, $\boldsymbol{v_g}(\boldsymbol{\mu}) \in \left[H^1(\Omega)\right]^d$ is the lifting vector field, with $\boldsymbol{v_g}(\boldsymbol{\mu})\big|_{\Gamma_D} = \boldsymbol{g}(\boldsymbol{\mu})$, $\boldsymbol{g}(\boldsymbol{x};\boldsymbol{\mu}) = \widetilde{\boldsymbol{g}}(\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu});\boldsymbol{\mu})$ being the velocity field prescribed on Γ_D. Then, the weak solution to (2.16) defined over the fixed domain Ω is given by $(\boldsymbol{v}(\boldsymbol{\mu}) + \boldsymbol{v_g}(\boldsymbol{\mu}), p(\boldsymbol{\mu}))$.

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

Recalling that in this work we confine the attention to two-dimensional variable shape domains, 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 [22] as a generalization of the well-known Gordon-Hall transfinite approach, aiming 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, $i = 1,\ldots,n$, the i-th edge of Ω, and by $\widetilde{\Gamma}_i(\boldsymbol{\mu}_g)$ the correspondent 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 $i = 1,\ldots,n$. 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 $\boldsymbol{x}_i$ (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 laying on the boundary $\Gamma = \partial\Omega$ of Ω. To this end, 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:

$$\boldsymbol{d}_i(t;\boldsymbol{\mu}_g) = \widetilde{\boldsymbol{\psi}}_i(t;\boldsymbol{\mu}_g) - \boldsymbol{\psi}_i(t), \quad \forall\, t\in[0,1], \forall\boldsymbol{\mu}_g\in\mathscr{P}_g. \qquad (2.37)$$

For each point on Γ_i, this function gives us the relative displacement between the new and the old position of the boundary [22]. 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 seen in the following, these functions are computed by solving just as many elliptic problems stated on Ω, i.e., *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, $i = 1,\ldots,n$, the weight function ξ_i acts as multiplier

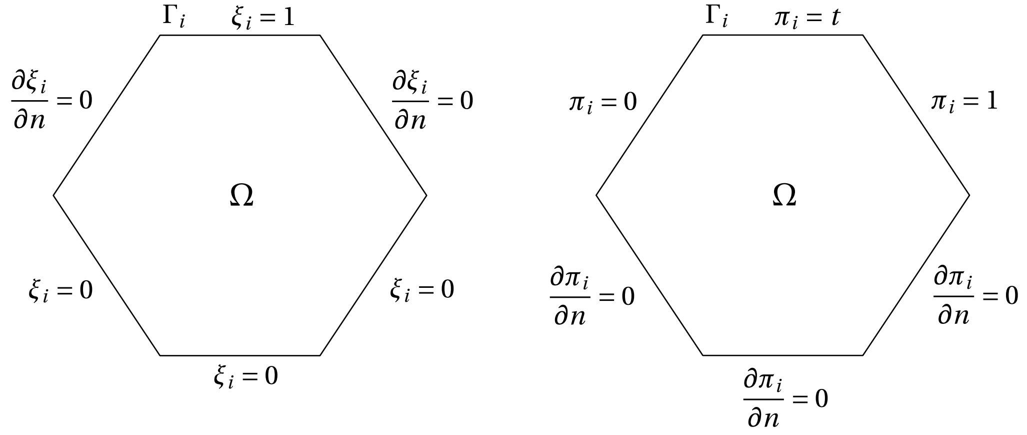
Γ_i
$\xi_i = 1$
$\frac{\partial \xi_i}{\partial n} = 0$
$\frac{\partial \xi_i}{\partial n} = 0$
Ω
$\xi_i = 0$
$\xi_i = 0$
$\xi_i = 0$
Γ_i
$\pi_i = t$
$\pi_i = 0$
$\pi_i = 1$
Ω
$\frac{\partial \pi_i}{\partial n} = 0$
$\frac{\partial \pi_i}{\partial n} = 0$
$\frac{\partial \pi_i}{\partial n} = 0$

Figure 2.1. Representation of the boundary conditions for the Laplace problems (2.38) (left) and (2.39) (right) stated on a exagonal reference domain Ω.

of $\boldsymbol{d}_i$ in the non-affine map, and it solves the following Laplace problem:

$$\begin{cases} \Delta \xi_i = 0 & \text{in } \Omega, \\ \xi_i = 1 & \text{on } \Gamma_i, \\ \dfrac{\partial \xi_i}{\partial n} = 0 & \text{on } \Gamma_j,\ j = i-1, i+1, \\ \xi_i = 0 & \text{on } \Gamma_j,\ j \neq i-1, i, i+1. \end{cases} \tag{2.38}$$

Here, $\partial/\partial n$ denotes the normal derivative, and we have implicitly assumed that if $i = n$, $\Gamma_{n+1} = \Gamma_1$ and if $i = 1$, $\Gamma_0 = \Gamma_n$. 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 Laplace problem over Ω, namely:

$$\begin{cases} \Delta \pi_i = 0 & \text{in } \Omega, \\ \pi_i = t & \text{on } \Gamma_i, \\ \pi_i = 0 & \text{on } \Gamma_{i-1}, \\ \pi_i = 1 & \text{on } \Gamma_{i+1}, \\ \dfrac{\partial \xi_i}{\partial n} = 0 & \text{on } \Gamma_j,\ j \neq i-1, i, i+1. \end{cases} \tag{2.39}$$

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 (2.38) and (2.39) are represented in Figure 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 (2.38) and (2.39) admit analytical solutions in closed form, which can be computed in a straightforward way without resorting to any numerical method. According to the edge ordering provided in Figure 2.2 and letting $\boldsymbol{x} = (x, y)$, we have:

$$\begin{aligned} &\xi_1 = 1 - y, \quad \xi_2 = 1 - x, \quad \xi_3 = y, \quad \xi_4 = x, \\ &\pi_1 = 1 - x, \quad \pi_2 = y, \quad \pi_3 = x, \quad \pi_4 = 1 - y. \end{aligned} \tag{2.40}$$

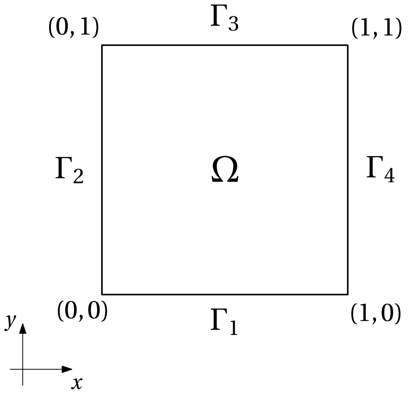

Figure 2.2. Clockwise enumeration for the edges of the reference squared domain Ω used in the numerical tests. The coordinates of the vertices are reported too.

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 [22]:

$$\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu}_g) = \boldsymbol{x} + \sum_{i=1}^{n} \left[\xi_i(\boldsymbol{x})\, \boldsymbol{d}_i(\pi_i(\boldsymbol{x};\boldsymbol{\mu}_g);\boldsymbol{\mu}) - \xi_i(\boldsymbol{x})\, \xi_{i+1}(\boldsymbol{x})\, \boldsymbol{d}_i(1;\boldsymbol{\mu}_g) \right], \tag{2.41}$$

with $\xi_{n+1} = \xi_1$. Observe that, for each edge Γ_i, $i = 1,\dots,n$, the associated displacement function is evaluated at the corresponding projection function π_i, which ranges between 0 and 1 due to the maximum principle applied to the elliptic boundary value problem (2.39) [42]. In our case, plugging (2.40) into (2.41) yields:

$$\begin{aligned} \boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu}_g) = \boldsymbol{x} &+ (1-y)\, \boldsymbol{d}_1(1-x;\boldsymbol{\mu}_g) - (1-y)\,(1-x)\, \boldsymbol{d}_1(1;\boldsymbol{\mu}_g) \\ &+ (1-x)\, \boldsymbol{d}_2(y;\boldsymbol{\mu}_g) - (1-x)\, y\, \boldsymbol{d}_2(1;\boldsymbol{\mu}_g) \\ &+ y\, \boldsymbol{d}_3(x;\boldsymbol{\mu}_g) - y\, x\, \boldsymbol{d}_3(1;\boldsymbol{\mu}_g) \\ &+ x\, \boldsymbol{d}_4(1-y;\boldsymbol{\mu}_g) - x\,(1-y)\, \boldsymbol{d}_4(1;\boldsymbol{\mu}_g). \end{aligned} \tag{2.42}$$

Let us conclude this section by highlighting a couple of relevant remarks on the BDD TM (2.41), 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 [22]. Moreover, in the numerical simulations we performed the map (2.42) 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 (2.42) 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 (2.20) 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 $\boldsymbol{\Phi}(\boldsymbol{\mu}_g)$. 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 (2.33), 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:

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

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., [5].

Conversely, to ensure the existence of a weak solution over Ω for the boundary value problem (2.16) for the Navier-Stokes equations, we simply require the domain Ω to be Lipschitz [38]. Whereas, uniqueness is guaranteed upon a *small data* hypothesis on the boundary conditions and a possible forcing term [10].

2.4 Finite element method

In this section, we address the discretization of the parametrized problem (2.21) 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., [37].

Let $V_h \subset V$ be a finite-dimensional subspace of V of dimension M. The finite FE approximation of the weak problem (2.20) can be cast in the form: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u_h(\boldsymbol{\mu}) \in V_h$ such that

$$g(u_h(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) = 0 \quad \forall v_h \in V_h. \tag{2.43}$$

The discretization (2.43) is known as *Galerkin approximation*, and therefore $u_h(\boldsymbol{\mu})$ is referred to as the *Galerkin solution* to the problem (2.20). To study the well-posedness of (2.43), 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 [42]

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

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 (2.43) 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

$$\gamma(\boldsymbol{\mu}) = \sup_{w_h \in V_h} \sup_{v_h \in V_h} \frac{dg[u_h(\boldsymbol{\mu})](w_h, v_h; \boldsymbol{\mu})}{\|w_h\|_V \, \|v_h\|_V} \leq \gamma_0 , \tag{2.44}$$

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

$$\beta(\boldsymbol{\mu}) = \inf_{w_h \in V_h} \sup_{v_h \in V_h} \frac{dg[u_h(\boldsymbol{\mu})](w_h, v_h; \boldsymbol{\mu})}{\|w_h\|_V \, \|v_h\|_V} \geq \beta_0 . \tag{2.45}$$

In the following, we generally assume that both (2.44) and (2.45) hold; yet in Section 2.4.2, we further investigate the implications of the latter requirement, known as *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 (2.43), one has to resort to some iterative method, e.g., the Newton's method. Starting from an initial guess $u_h^0(\boldsymbol{\mu})$ and up to convergence, 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

$$dg\big[u_h^k(\boldsymbol{\mu})\big](\delta u_h^k(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) = -g(u_h^k(\boldsymbol{\mu}), v_h; \boldsymbol{\mu}) \quad \forall v_h \in V_h \tag{2.46}$$

in the unknown $\delta u_h^k(\boldsymbol{\mu}) \in V_h$, 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.,

$$u_h(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu}) \, \phi_j(\boldsymbol{x}) . \tag{2.47}$$

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

$$g\left(\sum_{j=1}^{M} u_h^{(j)}(\boldsymbol{\mu}) \, \phi_j, \phi_i; \boldsymbol{\mu}\right) = 0 \quad \forall i = 1, \dots, M. \tag{2.48}$$

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

$$\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^M , \tag{2.49}$$

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

$$\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu})\big)_i = g\left(\sum_{j=1}^{M} u_j(\boldsymbol{\mu}) \, \phi_j, \phi_i; \boldsymbol{\mu}\right), \quad i = 1, \dots, M. \tag{2.50}$$

Then, for $k \geq 0$, the k-th iteration of the Newton's method applied to the system (2.49) entails the resolution of the *linear* system

$$\mathbb{J}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big) \, \delta\mathbf{u}_h^k(\boldsymbol{\mu}) = -\mathbf{G}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big), \quad \delta\mathbf{u}_h^k(\boldsymbol{\mu}) \in \mathbb{R}^M . \tag{2.51}$$

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 $D_uG(z,\boldsymbol{\mu})$ of $G(\cdot;\boldsymbol{\mu})$), $\mathbb{J}_h(\cdot; \boldsymbol{\mu})$ is defined as

$$\left(\mathbb{J}_h\big(\mathbf{u}_h^k(\boldsymbol{\mu}); \boldsymbol{\mu}\big)\right)_{i,j} = dg\big[u_h^k(\boldsymbol{\mu})\big](\phi_j, \phi_i; \boldsymbol{\mu}), \quad i, j = 1, \dots, M. \tag{2.52}$$

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 let $h = \max_{K\in\Omega_h} h_K$, thus motivating the subscript h used so far to decorate the discrete solution. Then, 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 [38]. Therefore, an approximation to the unknown field

$$u = (u_1, \dots, u_S) \in V = V_1 \times \ldots \times V_S \subset \left[H^1(\Omega)\right]^S \tag{2.53}$$

satisfying the differential problem (2.21) is sought in the discrete space

$$V_h = \left(V_1 \cap X_h^{r_1}\right) \times \ldots \times \left(V_S \cap X_h^{r_S}\right). \tag{2.54}$$

In other terms, we seek an approximation to the generic scalar variable u_k in a FE space properly modified to take the boundary conditions into account. It worths point out here 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.

To define a convenient basis for X_h^r, introduce a set of points $\{\boldsymbol{N}_i\}_{i=1}^M$, called *nodes*, which usually 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

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

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-point of each edge in the mesh.

Hereunder, we further detail the algebraic formulation of the FE discretization for the Poisson equation and the steady Navier-Stokes equations.

Algorithm 2.1 The Newton's method applied to the nonlinear system (2.49).

```
function $\mathbf{u}_h$ = 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 Equation (2.41)
    $k = 0$
    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
```

2.4.1 Nonlinear Poisson equation

Let $V_h = X_h^1 \cap H_{\Gamma_D}^1(\Omega)$. Recalling (2.34), the Frechét derivative $dg[z](\cdot, \cdot; \boldsymbol{\mu})$ of $g(\cdot, \cdot; \boldsymbol{\mu})$ is given by

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

with $k_r(\cdot, \cdot; \boldsymbol{\mu}) \, : \, \Omega \times \mathbb{R} \rightarrow \mathbb{R}$ being the partial derivative of $k(\cdot, \cdot; \boldsymbol{\mu})$ with respect to its second argment. Then, introducing the discrete shifting function $u_{g,h}(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{j=1}^n u_{g,h}^{(j)}(\boldsymbol{\mu}) \phi_j(\boldsymbol{x})$, obtained via, e.g., interpolation of the boundary conditions, for any $\boldsymbol{\mu} \in \mathscr{P}$ the residual vector $\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu})$ reads

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

with its Jacobian $\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu})$ given by

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

Observe that in (2.4.1) and (2.4.1) 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 generally needs a quadrature rule to compute these integrals, due to the nonlinearity of the diffusion coefficient k. These considerations clearly 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 condition (2.45) and thus retain the stability of the numerical method [39]. Since the differential form (2.16) is second order in the velocity and first order in the pressure, this suggests that the discretization of the velocity field should be somehow richer thant that one of the pressure field [37]. In this respect, 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 [35]. Therefore, according to the notation introduced in (2.53) and (2.54), we have

$$u = \left(v_x,\, v_y,\, p\right) \quad \text{and} \quad V_h = \left(X_h^2 \cap H^1_{\Gamma_D}(\Omega)\right) \times \left(X_h^2 \cap H^1_{\Gamma_D}(\Omega)\right) \times \left(X_h^1 \cap L^2(\Omega)\right). \tag{2.55}$$

Let further $\{\boldsymbol{V}_i\}_{i=1}^{M_v}$ be the set of vertices, associated with the linear FE basis functions $\{\phi_i^p\}_{i=1}^{M_v}$ used to discretize p, and $\{\boldsymbol{N}_i\}_{i=1}^{M_n}$ the set of nodes, associated with the quadratic FE basis functions $\{\phi_i^{\boldsymbol{v}}\}_{i=1}^{M_n}$ used to discretize $\boldsymbol{v}$. Then, setting $M = 2M_n + M_v$ and introducing the following 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, 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 $\{\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{2M_n}\}$ 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

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

Then, given (2.35), (2.36) and (2.50), the residual vector for the Navier-Stokes equations is defined as (we omit the dependence on $\boldsymbol{\mu}$ to ease the notation):

$$\begin{aligned}\big(\mathbf{G}_h(\mathbf{u}_h;\boldsymbol{\mu})\big)_i = \sum_{K\in\Omega_h}&\Bigg\{\nu\sum_{j=1}^{2Mn}\left[u_h^{(j)} + v_{\mathbf{g},h}^{(j)}\right]\int_K \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\boldsymbol{\phi}_j \;:\; \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\boldsymbol{\phi}_i\;|\mathbb{J}_{\boldsymbol{\Phi}}|\,dK\\ &+\sum_{j=1}^{2M_n}\sum_{l=1}^{2M_n}\left[u_h^{(j)}\;u_h^{(l)} + u_h^{(j)}\;v_{\mathbf{g},h}^{(l)} + v_{\mathbf{g},h}^{(j)}\;u_h^{(l)} + v_{\mathbf{g},h}^{(j)}\;v_{\mathbf{g},h}^{(l)}\right]\int_K\left(\boldsymbol{\phi}_j\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\right)\boldsymbol{\phi}_l\cdot\boldsymbol{\phi}_i\;|\mathbb{J}_{\boldsymbol{\Phi}}|\,dK\\ &-\frac{1}{\rho}\sum_{j=2M_n+1}^{M} u_h^{(j)}\int_K \phi_j^p\;\mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\cdot\boldsymbol{\phi}_i\;|\mathbb{J}_{\boldsymbol{\Phi}}|\,dK\Bigg\}, \qquad \text{for } i=1,\dots,2M_n,\\ \big(\mathbf{G}_h(\mathbf{u}_h;\boldsymbol{\mu})\big)_{2M_n+i} = \sum_{K\in\Omega_h}&\sum_{j=1}^{2M_n}\left[u_h^{(j)} + v_{\mathbf{g},h}^{(j)}\right]\int_K \mathbb{J}_{\boldsymbol{\Phi}}^{-T}\nabla\cdot\boldsymbol{\phi}_j\;\phi_i^p\;|\mathbb{J}_{\boldsymbol{\Phi}}|\,dK\Big\}, \qquad \text{for } i=1,\dots,M_v.\end{aligned}$$

Here, $\boldsymbol{v}_{\mathbf{g},h}(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{j=1}^{M_n}\left[v_{\mathbf{g},h}^{(j)}(\boldsymbol{\mu})\;\boldsymbol{\phi}_j(\boldsymbol{x}) + v_{\mathbf{g},h}^{(M_n+j)}(\boldsymbol{\mu})\;\boldsymbol{\phi}_{M_n+j}(\boldsymbol{x})\right]$ is the discrete lifting vector for the velocity field. Upon introducing the matrices $\mathbb{D}(\boldsymbol{\mu})\in\mathbb{R}^{2M_n\times 2M_n}$, $\mathbb{C}_i(\boldsymbol{\mu})\in\mathbb{R}^{2M_n\times 2M_n}$, $i=1,\dots,2M_n$, and $\mathbb{B}(\boldsymbol{\mu})\in\mathbb{R}^{M_v\times 2M_n}$, defined as

$$\big(\mathbb{D}(\boldsymbol{\mu})\big)_{i,j} = \sum_{K\in\Omega_h}\int_K \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\boldsymbol{\phi}_j \;:\; \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\boldsymbol{\phi}_i\;|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK, \qquad i,j=1,\dots,2M_n, \tag{2.56a}$$

$$\big(\mathbb{C}_i(\boldsymbol{\mu})\big)_{j,l} = \sum_{K\in\Omega_h}\int_K\left(\boldsymbol{\phi}_j\cdot\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\right)\boldsymbol{\phi}_l\cdot\boldsymbol{\phi}_i\;|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK, \qquad i,j,l=1,\dots,2M_n, \tag{2.56b}$$

$$\big(\mathbb{B}(\boldsymbol{\mu})\big)_{i,j} = \sum_{K\in\Omega_h}\int_k \phi_i^p\;\mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\cdot\boldsymbol{\phi}_j\;|\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})|\,dK, \qquad i=1,\dots,M_v,\; j=1,\dots,M_n, \tag{2.56c}$$

the components of the residual vector can be written in compact form as follows:

$$\begin{aligned}\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_i =&\,\nu(\boldsymbol{\mu})\big(\mathbb{D}(\boldsymbol{\mu})[\mathbf{v}_h(\boldsymbol{\mu})+\mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})]\big)_i\\ &+\mathbf{v}_h(\boldsymbol{\mu})^T\mathbb{C}_i(\boldsymbol{\mu})\mathbf{v}_h(\boldsymbol{\mu}) + \mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})^T\mathbb{C}_i(\boldsymbol{\mu})\mathbf{v}_h(\boldsymbol{\mu})\\ &+\mathbf{v}_h(\boldsymbol{\mu})^T\mathbb{C}_i(\boldsymbol{\mu})\mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu}) + \mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})^T\mathbb{C}_i(\boldsymbol{\mu})\mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})\\ &-\frac{1}{\rho(\boldsymbol{\mu})}\big(\mathbb{B}^T(\boldsymbol{\mu})\mathbf{p}_h\big)_i, \qquad i=1,\dots,2M_n,\end{aligned}\tag{2.57a}$$

$$\big(\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_{2M_n+i} = \big(\mathbb{B}(\boldsymbol{\mu})[\mathbf{v}_h(\boldsymbol{\mu})+\mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})]\big)_i, \qquad i=1,\dots,M_v. \tag{2.57b}$$

Hence, the Jacobian is given by:

$$\big(\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_{i,j} = \begin{cases}\nu(\boldsymbol{\mu})\;\big(\mathbb{D}(\boldsymbol{\mu})\big)_{i,j} + \Big(\big(\mathbb{C}_i(\boldsymbol{\mu})+\mathbb{C}_i^T(\boldsymbol{\mu})\big)\big(\mathbf{v}_h(\boldsymbol{\mu})+\mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})\big)\Big)_j & i,j=1,\dots,2M_n,\\ -\dfrac{1}{\rho(\boldsymbol{\mu})}\big(\mathbb{B}^T(\boldsymbol{\mu})\big)_{i,j-2M_n} & i=1,\dots,2M_n,\; j=2M_n+1,\dots,M,\\ \big(\mathbb{B}(\boldsymbol{\mu})\big)_{i,j} & i=1,\dots,M_v,\; j=1,\dots,2M_n,\\ 0 & i,j=2M_n+1,\dots,M,\end{cases}$$

or, in matrix form:

$$\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu}) = \begin{bmatrix} \nu(\boldsymbol{\mu})\mathbb{D}(\boldsymbol{\mu}) + \mathbb{C}(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu}) & -\dfrac{1}{\rho(\boldsymbol{\mu})}\mathbb{B}^T(\boldsymbol{\mu}) \\ \mathbb{B}(\boldsymbol{\mu}) & 0 \end{bmatrix}, \tag{2.58}$$

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

$$\big(\mathbb{C}(\mathbf{u}_h(\boldsymbol{\mu});\boldsymbol{\mu})\big)_{i,j} = \Big(\big(\mathbb{C}_i(\boldsymbol{\mu}) + \mathbb{C}_i^T(\boldsymbol{\mu})\big)\big(\mathbf{v}_h(\boldsymbol{\mu}) + \mathbf{v}_{\mathbf{g},h}(\boldsymbol{\mu})\big)\Big)_j \qquad \text{for } i, j = 1, \ldots, 2M_n.$$

2.5 POD-Galerkin reduced basis method

As detailed in the previous section, the finite element discretization of the $\boldsymbol{\mu}$-dependent nonlinear differential problem (2.21), combined with the Newton's method, entails the assembly and resolution of (possibly) many linear systems of the form (2.51), 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 $\boldsymbol{\mu} \in \mathscr{P}$ [9]. This motivates the broad and continuous spread of *reduced-order* models, accross several inter-disciplinary areas, e.g., parameter estimation, optimal control, shape optimization and uncertainty quantification [19, 38].

As widely anticipated, in this work we focus on reduced basis (RB) methods for the approximation of the variational problem (2.21). In this respect, 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 correspondent continuous solution $u_h(\boldsymbol{\mu})$ (either refining the computational mesh or 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 (2.21) as a linear combination of an ensemble of parameter-independent functions $\{\psi_1, \ldots, \psi_L\} \subset V_h$, called *reduced basis functions*, built from a collection of high-fidelity solutions $\{u_h(\boldsymbol{\mu}^{(1)}), \ldots, u_h(\boldsymbol{\mu}^{(N)})\} \subset \mathscr{M}_h$, called *snapshots*, where the discrete and finite set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \ldots, \boldsymbol{\mu}^{(N)}\} \subset \mathscr{P}$ may consist of either a uniform lattice or randomly generated points over the parameter domain $\mathscr{P}$ [19]. 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., [6]). Here, we pursue the first approach, employing the well-known Proper Orthogonal Decomposition (POD) method [44], 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* $u_{rb}(\boldsymbol{\mu})$ is sought in the form

$$u_{\mathtt{rb}}(\boldsymbol{x}; \boldsymbol{\mu}) = \sum_{l=1}^{L} u_{\mathtt{rb}}^{(l)}(\boldsymbol{\mu})\ \psi_l(\boldsymbol{x}) \in V_{\mathtt{rb}}, \tag{2.59}$$

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 easily foresee, RB methods (potentially) enable a relevant reduction in the computational effort when the associated test and trial space $V_{\mathtt{rb}}$ features a dimensionality which is significantly smaller than the dimension of the original finite element space V_h; in other terms, $L << M$ must hold. However, this tacitly assumes that the solution manifold $\mathscr{M} \subset V$ (or, equivantely, $\mathscr{M}_h \subset V_h$, for what previously said) is actually of low-dimension, and can then be accurately approximated by a subspace of reduced dimension L [19]. To further investigate this necessary hypothesis, it worths introduce the notion of Kolmogorov L-width. Here, we report the definition provided in [27].

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*

$$E(A; X_L) = \sup_{x \in A} \inf_{y \in X_L} \left\|x - y\right\|_X. \tag{2.60}$$

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

$$\begin{aligned} d_L(A, X) &= \inf\{E(A; X_L)\ :\ X_L \textit{ is an } L\textit{-dimensional subspace of } X\} \\ &= \inf_{X_L} \sup_{x \in A} \inf_{y \in X_L} \left\|x - y\right\|_X. \end{aligned} \tag{2.61}$$

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 [27]. 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 [3].

Nevertheless, we still have to ensure that the RB space $V_{\mathtt{rb}}$, i.e., the chosen X_L in (2.61), actually attains the infimum $d_L(\mathscr{M}, V)$, or at least a value close to the infimum. In this respect, let us consider the following bound for the error committed when approximating the continuous solution $u(\boldsymbol{\mu})$ with $u_{\mathtt{rb}}(\boldsymbol{\mu})$:

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

The first term on the right-hand side measures the discrepancy between the continuous solution and its high-fidelity approximation provided by the FE method (or any discretization method of choice, e.g., finite difference or spectral methods). 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 term $\left\| u_h(\boldsymbol{\mu}) - u_{\mathtt{rb}}(\boldsymbol{\mu}) \right\|_V$, i.e., the error between the truth and the reduced solution. In this respect, the last decade has whitnessed the development of different *a priori* and *a posteriori* estimates for such an error (see, e.g., [3, 19, 27]), thus *certifying* the RB procedure, that is, enabling the user to trust the output of the RB method. However, as already mentioned in the introduction to the chapter, the range of application of these estimates is usually limited to linear problems with an affine dependence on the parameter. Although recent and relevant improvements have been achieved also for the Navier-Stokes equations (see, e.g., [9, 38]), they rely on non-trivial results from functional analysis and involve rather long calculations. Hence, it is not 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 $\left\| u_h(\boldsymbol{\mu}) - u_{\mathtt{rb}}(\boldsymbol{\mu}) \right\|_V$ on a finite and discrete *test* dataset $\Xi_{te} \in \mathscr{P}$. We refer the reader to Section 2.6 for a keener discussion.

Let us now resume the derivation of the POD-Galerkin RB method. To unearth $u_{\mathtt{rb}}(\boldsymbol{\mu})$, whose general form is given in (2.59), we proceed to project the variational problem (2.21) onto the RB space $V_{\mathtt{rb}}$ pursuing a stanard Galerkin approach, leading to the following *reduced basis problem*: given $\boldsymbol{\mu} \in \mathscr{P}$, find $u_{\mathtt{rb}}(\boldsymbol{\mu}) \in V_{\mathtt{rb}}$ so that

$$g(u_{\mathtt{rb}}(\boldsymbol{\mu}), v_{\mathtt{rb}}; \boldsymbol{\mu}) = 0 \quad \forall v_{\mathtt{rb}} \in V_{\mathtt{rb}} . \tag{2.62}$$

Then, the Newton's method applied to (2.62) entails, at each iteration $k \geq 0$, the resolution of the following linearized problem: given $\boldsymbol{\mu} \in \mathscr{P}$, seek $\delta u_{\mathtt{rb}}^k(\boldsymbol{\mu})$ such that

$$dg\left[u_{\mathtt{rb}}^k(\boldsymbol{\mu})\right]\left(\delta u_{\mathtt{rb}}^k(\boldsymbol{\mu}), v_{\mathtt{rb}}; \boldsymbol{\mu}\right) = -g\left(u_{\mathtt{rb}}^k(\boldsymbol{\mu}), v_{\mathtt{rb}}; \boldsymbol{\mu}\right) \quad \forall v_{\mathtt{rb}} \in V_{\mathtt{rb}} , \tag{2.63}$$

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

Moving towards an algebraic standpoint, 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 $\mathbb{V} = \left[\boldsymbol{\psi}_1, \middle| \dots \middle|, \boldsymbol{\psi}_L\right] \in \mathbb{R}^{M \times L}$. For any $v_{\mathtt{rb}} \in V_{\mathtt{rb}}$, the matrix $\mathbb{V}$ encodes the change of variables from the RB basis to the standard FE basis, i.e.,

$$\mathbf{v}_L = \mathbb{V}\, \mathbf{v}_{\mathtt{rb}} . \tag{2.64}$$

Therefore, each element $v_{\mathtt{rb}}$ of the reduced space admits two (algebraic) representations:

- $\mathbf{v}_{\mathtt{rb}} \in \mathbb{R}^L$, collecting the coefficients for the expansion of $v_{\mathtt{rb}}$ 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 $v_{\mathtt{rb}}$ 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 $v_{\mathtt{rb}} = \psi_l$, $l = 1, \dots, L$, in the RB problem (2.62), for any $\boldsymbol{\mu} \in \mathscr{P}$ we get the following set of equations:

$$g(u_{\mathtt{rb}}(\boldsymbol{\mu}), \psi_l; \boldsymbol{\mu}) = 0 \quad 1 \le l \le L. \tag{2.65}$$

Plugging into (2.65) 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:

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

where the last equality follows from the definition (2.50) of the residual vector $\mathbf{G}_h(\cdot; \boldsymbol{\mu})$ and the notation introduced in (2.64). Then, the algebraic formulation of the reduced basis problem (2.62) can be written in compact form as:

$$\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbb{V} \, \mathbf{u}_{\mathtt{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L. \tag{2.66}$$

Hence, the *reduced nonlinear system* (2.66) 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

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{w}; \boldsymbol{\mu}) = \mathbb{V}^T \mathbb{J}_h(\mathbb{V} \, \mathbf{w}; \boldsymbol{\mu}) \mathbb{V} \in \mathbb{R}^{L \times L} \quad \forall \mathbf{w} \in \mathbb{R}^L, \forall \boldsymbol{\mu} \in \mathscr{P}, \tag{2.67}$$

so that, starting from an initial guess $\mathbf{u}_{\mathtt{rb}}^0 \in \mathbb{R}^L$, each iteration k, $k \ge 0$, of the Newton's method applied to the reduced nonlinear system (2.66) entails the resolution of the linear system

$$\mathbb{J}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu}); \boldsymbol{\mu}) \, \delta\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu}) = -\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu}); \boldsymbol{\mu}), \tag{2.68}$$

with $\mathbf{u}_{\mathtt{rb}}^{k+1}(\boldsymbol{\mu}) = \mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu}) + \delta\mathbf{u}_{\mathtt{rb}}^k(\boldsymbol{\mu})$. Therefore, as previously anticipated, resorting to 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 [38]. Therefore, a meaningful low-dimensional representation of data is obtained by truncating

Algorithm 2.2 The Newton's method applied to the reduced nonlinear system (2.66).

```
function $\mathbf{u}_{\mathtt{rb}} = \textsc{NewtonRB}(\boldsymbol{\mu},\ \Omega_h,\ \mathbb{V},\ \mathbf{u}^0_{\mathtt{rb}},\ \{\xi_i\}_{i=1}^n,\ \{\pi_i\}_{i=1}^n,\ \delta,\ K_{max})$
    build the transformation map $\boldsymbol{\Phi}(\boldsymbol{x};\boldsymbol{\mu})$ via Equation (2.41)
    $k = 0$
    do
        set $\mathbf{u}_L^k = \mathbb{V}\mathbf{u}_{\mathtt{rb}}^k$
        evaluate the reduced residual vector $\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu}) = \mathbb{V}^T\mathbf{G}_h(\mathbf{u}_L^k;\boldsymbol{\mu})$
        form the Jacobian $\mathbb{J}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu}) = \mathbb{V}^T\mathbb{J}_h(\mathbf{u}_L^k;\boldsymbol{\mu})\mathbb{V}$
        solve $\mathbb{J}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu})\,\delta\mathbf{u}_{\mathtt{rb}}^k = -\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k;\boldsymbol{\mu})$
        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 $\|\mathbf{G}_{\mathtt{rb}}(\mathbf{u}_{\mathtt{rb}}^k)\| > \delta$
    $\mathbf{u}_{\mathtt{rb}} = \mathbf{u}_{\mathtt{rb}}^k$
end function
```

the orthonormal basis to retain only a few POD modes, then projecting the system onto the truncated basis [44]. As a matter of fact, 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 [17].

Let us first precisely define our objective. Consider a collection of N snapshots (i.e., high-fidelity solutions to the problem (2.21)) $\{u_h(\boldsymbol{\mu}^{(1)}),\dots,u_h(\boldsymbol{\mu}^{(N)})\} \subset \mathcal{M}_h$ correspondent to the finite and discrete parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)},\dots,\boldsymbol{\mu}^{(N)}\} \subset \mathscr{P}$, and let $\mathcal{M}_{\Xi_N}$ be the subspace spanned by the snapshots, i.e.,

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

Clearly, $\mathcal{M}_{\Xi_N} \subset \mathcal{M}_h$ and we can assume that $\mathcal{M}_{\Xi_N}$ provides a good approximation of $\mathcal{M}_h$, 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 $\mathcal{M}_{\Xi_N}$, i.e., a collection of FE functions $\{\psi_1,\dots,\psi_L\} \subset \mathcal{M}_{\Xi_N}$, with $L \ll M$, N, and L *independent of* M and N, so that the associated linear space

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

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

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

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

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

$$\mathbb{S} = \mathbb{W}\begin{bmatrix} \mathbb{D} & \mathbb{O}_{R\times(N-R)} \\ \mathbb{O}_{(M-R)\times R} & \mathbb{O}_{(M-R)\times(N-R)} \end{bmatrix}\mathbb{Z}^T = \mathbb{W}\Sigma\mathbb{Z}^T, \tag{2.69}$$

where $\mathbb{O}_{p\times q}$ denotes the null matrix with p rows and q columns. The real values $\{\sigma_i\}_{i=1}^R$ are called *singular values* of $\mathbb{S}$, the columns $\{\mathbf{w}_m\}_{m=1}^M$ of $\mathbb{W}$ are called *left singular vectors* of $\mathbb{U}$, and the columns $\{\mathbf{w}_n\}_{n=1}^N$ of $\mathbb{Z}$ are called *right singular vectors* of $\mathbb{U}$, and they are linked by the following relations:

$$\mathbb{S}\mathbb{S}^T\mathbf{w}_m = \begin{cases} \sigma_m^2\,\mathbf{w}_m & \text{for } 1 \le m \le R, \\ \mathbf{0} & \text{for } R+1 \le m \le M, \end{cases} \tag{2.70}$$

$$\mathbb{S}^T\mathbb{S}\,\mathbf{z}_n = \begin{cases} \sigma_n^2\,\mathbf{z}_n & \text{for } 1 \le n \le R, \\ \mathbf{0} & \text{for } R+1 \le n \le N, \end{cases} \tag{2.71}$$

$$\mathbb{S}\,\mathbf{z}_i = \sigma_i\,\mathbf{w}_i \qquad \text{for } 1 \le i \le R, \tag{2.72}$$

$$\mathbb{S}^T\mathbf{w}_i = \sigma_i\,\mathbf{z}_i \qquad \text{for } 1 \le i \le R. \tag{2.73}$$

In particular, Equations (2.70) and (2.71) 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 Equation (2.69), the SVD of $\mathbb{S}$ can be cast in the compact form:

$$\mathbb{S} = \mathbb{W}^R\mathbb{D}\big(\mathbb{Z}^R\big)^T, \tag{2.74}$$

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.,

$$\mathbb{W}^R = \big[\mathbf{w}_1 \,\big|\, \dots \,\big|\, \mathbf{w}_R\big] \quad \text{and} \quad \mathbb{Z}^R = \big[\mathbf{z}_1 \,\big|\, \dots \,\big|\, \mathbf{z}_R\big]. \tag{2.75}$$

Letting $\mathbb{B}^R = \mathbb{D}\big(\mathbb{Z}^R\big)^T \in \mathbb{R}^{R\times N}$ and exploiting the orthonormality of the columns of $\mathbb{W}^R$, the generic column $\mathbf{s}_n = \mathbf{u}_h\big(\boldsymbol{\mu}^{(n)}\big)$ of $\mathbb{S}$, $n = 1,\dots N$, can be expressed as [44]:

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

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

Assume now we want to approximate the columns of $\mathbb{S}$ by means of L orthonormal vectors $\{\widetilde{\mathbf{w}}_1,\dots,\widetilde{\mathbf{w}}_L\}$, with $L < R$. It is an easy matter to show that for each $\mathbf{s}_n$, $n = 1,\dots,N$, the element of $\mathrm{span}\{\widetilde{\mathbf{w}}_1,\dots,\widetilde{\mathbf{w}}_L\}$ closest to $\mathbf{s}_n$ in the Euclidean norm $\|\cdot\|_{\mathbb{R}^M}$ is given by

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

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

$$\varepsilon(\widetilde{\mathbf{w}}_1,\dots,\widetilde{\mathbf{w}}_L) = \sum_{n=1}^N \left\| \mathbf{s}_n - \sum_{l=1}^L \big(\mathbf{s}_n, \widetilde{\mathbf{w}}_l\big)_{\mathbb{R}^M}\widetilde{\mathbf{w}}_l \right\|^2_{\mathbb{R}^M}. \tag{2.76}$$

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 (2.76) among all the orthonormal bases of $\mathbb{R}^L$.

Theorem 2.5.1 (Schmidt-Eckart-Young)**.** *Consider a rectangular matrix* $\mathbb{S} = [\mathbf{s}_1 \,|\, \dots \,|\, \mathbf{s}_N] \in \mathbb{R}^{M\times N}$ *with rank* $R \leq \min\{M, N\}$*, and let* $\mathbb{S} = \mathbb{W}\Sigma\mathbb{Z}^T$ *be the singular value decomposition (SVD) of* $\mathbb{S}$*, with* $\mathbb{W} = [\mathbf{w}_1 \,|\, \dots \,|\, \mathbf{w}_M] \in \mathbb{R}^{M\times M}$ *and* $\mathbb{Z} = [\mathbf{z}_1 \,|\, \dots \,|\, \mathbf{z}_N] \in \mathbb{R}^{N\times N}$ *orthogonal matrices, and* $\Sigma \in \mathbb{R}^{M\times N}$ *defined as in* (2.69)*. Further, let* $\mathbb{S} = \mathbb{W}^R\mathbb{D}\big(\mathbb{Z}^R\big)^T = \mathbb{W}^R\mathbb{B}^R$ *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* (2.75)*,* $\mathbb{D} = \mathrm{diag}(\sigma_1, \dots, \sigma_R) \in \mathbb{R}^{R\times R}$*, and* $\mathbb{B}^R = \mathbb{D}\big(\mathbb{Z}^R\big)^T \in \mathbb{R}^{R\times N}$.
Suppose that $\widehat{\mathbb{W}}^R \in \mathbb{R}^{M\times R}$ *denotes a matrix with pairwise orthonormal vectors* $\widehat{\mathbf{w}}_r$*,* $r = 1, \dots, R$*, 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*

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

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

$$\big\|\mathbb{S} - \mathbb{W}^L\mathbb{B}^L\big\|_F \leq \big\|\mathbb{S} - \widehat{\mathbb{W}}^L\mathbb{C}^L\big\|_F. \tag{2.77}$$

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

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

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.,* $\mathbb{C}$*) [44].*

Regarding (2.77), let us note that

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

Then, according to (2.76),

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

and, analogously,

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

Hence, the optimality condition (2.77) actually implies

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

for any set $\{\widehat{\mathbf{w}}_l\}_{l=1}^L$ of pairwise orthonormal vectors. Moreover, it can be shown that (see, e.g., [44])

$$\varepsilon(\mathbf{w}_1,\dots,\mathbf{w}_L) = \sum_{j=L+1}^{R} \sigma_j^2, \tag{2.78}$$

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 *POD basis* of rank L. Coming back to the POD-Galerkin RB method, we set $\boldsymbol{\psi}_l = \mathbf{w}_l$, for all $l = 1,\dots,L$, so that

$$\mathbb{V} = \left[\mathbf{w}_1 \,\middle|\, \dots \,\middle|\, \mathbf{w}_L\right]. \tag{2.79}$$

Hence, in the reduced basis problem (2.62) the test and trial functions are picked from the subspace $V_{\texttt{rb}} = V_{\mathrm{POD}}$ of V_h, spanned by the FE functions $\{\psi_l\}_{l=1}^L$, given by

$$\psi_l(\boldsymbol{x}) = \sum_{m=1}^{M} \psi_l^{(m)}\, \phi_m(\boldsymbol{x}) = \sum_{m=1}^{M} (\mathbf{w}_l)_m\, \phi_m(\boldsymbol{x}) \quad \text{for } 1 \le l \le L. \tag{2.80}$$

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 $(\cdot,\cdot)_h$:

$$(\chi_h,\xi_h)_h = \sum_{i=1}^{M} \chi_h(\boldsymbol{N}_i)\,\xi_h(\boldsymbol{N}_i) = \sum_{i=1}^{M} \chi_h^{(i)}\,\xi_h^{(i)} = (\boldsymbol{\chi}_h,\boldsymbol{\xi}_h)_{\mathbb{R}^M}. \tag{2.81}$$

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

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

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

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

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

then by (2.72) we have

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

Let us conclude by further discussing the estimate (2.78) 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 [19]. 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 [4]. 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:

$$\frac{\sum_{l=1}^{L}\sigma_l^2}{\sum_{l=1}^{R}\sigma_l^2} > 1-\delta^2. \tag{2.82}$$

In other terms, the energy retained by the first L POD modes must be greater than the fraction $1-\delta^2$ of the total energy of the snapshots [38]. This criterium, also known as *relative information content*, has been embodied into Algorithm 2.3 summarizing the POD method. Moreover, it also implies that

$$\frac{\left\| \mathbb{S} - \mathbb{W}^L \mathbb{B}^L \right\|_F}{\| \mathbb{S} \|_F} < \delta \, .$$

However, (2.78), then (2.82), only holds for the columns of $\mathbb{S}$, i.e., for the snapshots. While it can be readily generalized to any element in the snapshot manifold $\mathscr{M}_{\Xi_N}$[5], 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 (2.78) on a discrete and finite parameter test set $\Xi_{te} \subset \mathscr{P}$, with $\Xi_{te} \cap \Xi_N = \emptyset$.

Algorithm 2.3 The POD algorithm.

```
function $\mathbb{V}$ = POD($\mathbb{S}$, $\delta$)
    if $M \le N$ 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 $\mathbb{K}\mathbf{z}_i = \lambda_i \, \mathbf{z}_i$
            $\mathbf{w}_i = \frac{1}{\sqrt{\lambda_i}} \mathbb{S} \, \mathbf{z}_i$
        end for
    end if
    find the minimum $L$ satisfying (2.82)
    $\mathbb{V} = [\mathbf{w}_1 \,|\, \dots \,|\, \mathbf{w}_L]$
end function
```

2.5.2 Implementation: details and issues

The whole numerical procedure presented so far can be efficiently carried out within an offline-online framework [36]. The parameter-independent *offline* step consists of the

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

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

generation of the snapshots through a high-fidelity, expensive discretization method (e.g., the finite element method) and the subsequent construction of the reduced basis through the POD technique. Then, given a new parameter value $\boldsymbol{\mu} \in \mathscr{P}$, the nonlinear reduced system (2.66) is solved *online* resorting to, e.g., the Newton's method, which entails the assembly and resolution of linear systems of the form (2.68). The main steps of the resulting POD-Galerkin (POD-G) RB method are summarized in Algorithm 2.4.

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

```
function $[\mathbb{V}, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n] = \text{PODG}\textsc{offline}(\mathscr{P}, \Omega_h, N, \delta_{\mathtt{POD}}, \delta_{\mathtt{NWT}}, K_{max})$
    solve the Laplace problems (2.38) and (2.39), yielding $\{\xi_i\}_{i=1}^n$ and $\{\pi_i\}_{i=1}^n$
    generate the parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$
    $\mathbf{u}_h(\boldsymbol{\mu}^{(i)}) = \text{Newton}(\boldsymbol{\mu}^{(i)}, \mathbf{u}_h^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\mathtt{NWT}}, K_{max})$,  for $i = 1, \dots, N$
    $\mathbb{S} = [\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \,|\, \dots \,|\, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})]$
    $\mathbb{V} = \text{POD}(\mathbb{S}, \delta_{\mathtt{POD}})$
end function

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

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. In this respect, notice that the operative definitions (2.66) and (2.67) of the reduced residual vector $\mathbf{G}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$ and its Jacobian $\mathbb{J}_{\mathtt{rb}}(\cdot;\boldsymbol{\mu})$, respectively, 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 is clearly dependent on M. On the other hand, 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 (2.68) has to be embodied directly in the online stage, thus seriously compromising the efficiency of the overall procedure [2]. Without escaping the algebraic framework, this can be successfully overcome upon resorting to suitable techniques as the discrete empirical interpolation method (DEIM) or its matrix variant (MDEIM) [31]. 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

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

and $\alpha_j^q(\cdot;\cdot)$, $q = 1, \dots, Q_j$, solution- and parameter-dependent scalar coefficients [38]. Notice that this approach guarantees a sensible 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 $\alpha_g^j(\cdot;\cdot)$.

Anyway, in this work we shall not pursue this method. Rather, we propose an alternative, non-algebraic way to tackle any online query which completely skip the assembly and resolution of the reduced system. We refer the reader to Section **??** 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.

Application to the steady Navier-Stokes equations

TODO

2.6 A POD-based RB method using neural networks

Although intuitive, as resulting from a straightforward projection procedure, a finer analysis may reveal a few drawbacks for the standard POD-Galerkin reduced basis technique, particularly for nonlinear differential problems featuring a nonaffine dependence on the parameters. In those case, 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 remarkable 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 could be cumbersome. 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 to guarantee the same accuracy provided by the standard POD-Galerkin method [2].

Unfortunately, the disadvantages of a projection-based RB method are not confined to mere implementative aspects. In this respect, 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 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 should worth remark that there exists a one-to-onde correspondence between $V_{\mathtt{rb}}$ and 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 Equation (2.80) follows:

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

In particular, this implies that the projection of any $v_h \in V_h$ onto $V_{\mathtt{rb}}$ in the discrete scalar product $(\cdot,\cdot)_h$ (see Equation (2.81)) algebraically corresponds to the projection of $\mathbf{v}_h$ onto Col($\mathbb{V}$), given by

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

Noting that $\mathbb{P}\, \mathbf{v}_h$ is the element of Col($\mathbb{V}$) closest to $\mathbf{v}_h$ in the Euclidean norm, one could reasonably expect that the solution to the reduced system coincides with the projection of the (unknown) high-fidelity solution $\mathbf{u}_h$ onto Col($\mathbb{V}$), i.e.,

$$\mathbf{u}_L = \mathbb{P}\, \mathbf{u}_h = \mathbb{V}\mathbb{V}^T \mathbf{u}_h\,,$$

or equivalently

$$\mathbf{u}_{\mathtt{rb}} = \mathbb{V}^T \mathbf{u}_h\,.$$

However, there exist many problems, even linear, for which the above equalities do not hold, i.e., for which the RB solution does not represent the best approximation we can construct within $V_{\mathtt{rb}}$ (or equivalently Col($\mathbb{V}$)). To get a sense of this, let us briefly consider the following boundary value problem for the linear two-dimensional Poisson equation:

$$\begin{cases} -\widetilde{\Delta}\widetilde{u}(\boldsymbol{\mu}) = \widetilde{f}(\widetilde{x}, \widetilde{y}) = 2\sin(\widetilde{x})\cos(\widetilde{y}) & \text{in } \widetilde{\Omega}(\boldsymbol{\mu})\,, \\ \widetilde{u}(\boldsymbol{\mu}) = \sin(\widetilde{\sigma}_x)\cos(\widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu})\,. \end{cases} \tag{2.83}$$

where $\widetilde{\Omega}(\boldsymbol{\mu})$ is the quadrilateral domain shown on the left in Figure 2.3, with $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \left[\pi/4, 3\pi/4\right] \times \left[0, \pi/2\right]$. This test case will be extensively treated in Section **??**; here, we are just interested in analyzing the errors $\left\|\mathbf{u}_h - \mathbb{V}\mathbb{V}^T\mathbf{u}_h\right\|_{\mathbb{R}^M}$ and $\|\mathbf{u}_h - \mathbf{u}_L\|_{\mathbb{R}^M}$, normalized by $\|\mathbf{u}_h\|_{\mathbb{R}^M}$[6], for different values of L. To this end, consider the right plot of Figure 2.3, reporting the average values of the aforementioned errors evaluated on a test dataset $\Xi_{te} \in \mathscr{P}$ consisting of $N_{te} = 200$ randomly picked samples. The POD basis has been constructed based on $N = 100$ snapshots, with the parameter set Ξ_N generated through the Latin Hypercube Sampling [21]. Note that in this case, the reduced *linear* system reads:

$$\mathbb{V}^T \mathbb{A}(\boldsymbol{\mu})\mathbb{V} = \mathbb{V}^T \mathbf{b}(\boldsymbol{\mu})\,,$$

with the stiffness matrix $\mathbb{A}(\boldsymbol{\mu}) \in \mathbb{R}^{M\times M}$ and the vector $\mathbf{b}(\boldsymbol{\mu}) \in \mathbb{R}^M$ defined as

$$\begin{aligned} \left(\mathbb{A}(\boldsymbol{\mu})\right)_{i,j} &= \int_\Omega \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_j \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})| d\Omega\,, \\ \left(\mathbf{b}(\boldsymbol{\mu})\right)_i &= \int_\Omega f\, \phi_i\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})| d\Omega - \int_\Omega \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla u_g(\boldsymbol{\mu}) \cdot \mathbb{J}_{\boldsymbol{\Phi}}^{-T}(\boldsymbol{\mu})\nabla\phi_i\, |\mathbb{J}_{\boldsymbol{\Phi}}(\boldsymbol{\mu})| d\Omega\,, \end{aligned}$$

where $u_g(x, y) = \sin(x)\cos(y)$ is the lifting vector and $\widetilde{u}_g(\boldsymbol{x}; \boldsymbol{\mu}) = \left(u_g \circ \boldsymbol{\Phi}(\boldsymbol{\mu})\right)(\boldsymbol{x})$. Although featuring an exponential decay, the error yielded by the POD-Galerkin method decreases

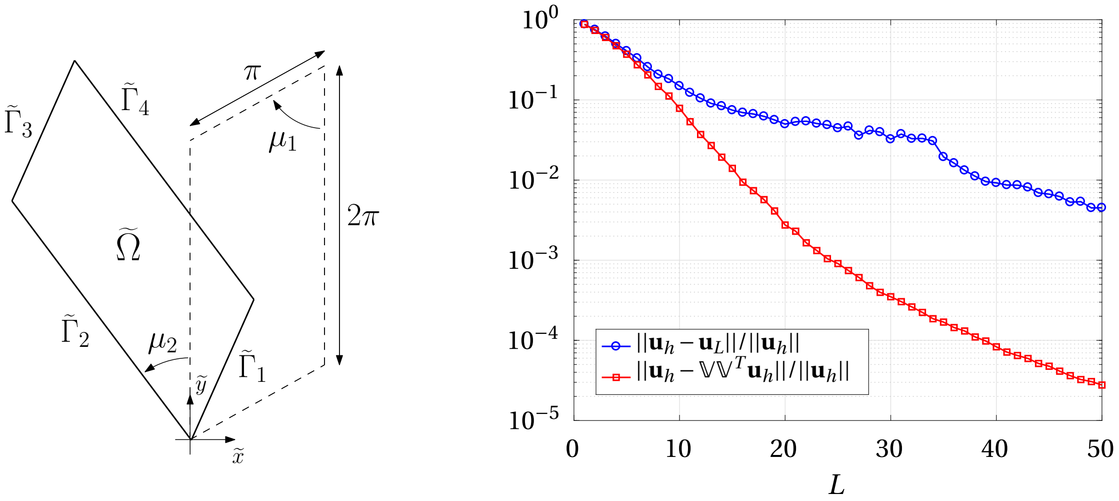

Figure 2.3. Left: computational domain $\widetilde{\Omega} = \widetilde{\Omega}(\boldsymbol{\mu})$ (solid line) for the linear Poisson problem (2.83). Right: average relative error committed by approximating the FE solution either through its projection onto the reduced space (red) or the POD-Galerkin RB solution (blue); the errors have been evaluated on a test parameter set $\Xi_{te} \subset \mathscr{P}$ consisting of $N_{te} = 200$ randomly picked values.

more slowly than the projection error does, and they may differ significantly. For instance, with $L = 50$ basis functions the two errors differ by two orders of magnitude.

Clearly, one can not expect a nicer scenario in the nonlinear case. Although it really depends on the type of nonlinearity featured by the PDE and the transformation map $\boldsymbol{\Phi}(\boldsymbol{\mu})$, nonlinear solvers (e.g., the Newton's method) may converge dramatically slowly, or even fail to converge, providing a solution $\mathbf{u}_{\mathtt{rb}}$ which is completely off with respect to $\mathbb{V}^T\mathbf{u}_h$ (i.e., our target). This could be ascribed to the fullness of the reduced Jacobian $\mathbb{J}_{\mathtt{rb}}(\mathbf{w}_{\mathtt{rb}}; \boldsymbol{\mu}) = \mathbb{V}^T \mathbb{J}_h(\mathbb{V}\, \mathbf{w}_{\mathtt{rb}}; \boldsymbol{\mu})\mathbb{V}$, as opposed to the sparsity of $\mathbb{J}_h(\cdot; \boldsymbol{\mu})$, resulting from the local nature of the FE basis functions. In turn, the fullness of $\mathbb{J}_{\mathtt{rb}}$, following from the fullness of $\mathbb{V}$, leads to a craggy surface, thus obstructing the convergence of the solver. In this respect, consider the following nonlinear Poisson problem:

$$\begin{cases} -\widetilde{\nabla} \cdot \left(\exp\left(\widetilde{u}(\boldsymbol{\mu})\right)\widetilde{\nabla}\widetilde{u}(\boldsymbol{\mu})\right) = \widetilde{f} & \text{in } \widetilde{\Omega}(\boldsymbol{\mu}), \\ \widetilde{u}(\boldsymbol{\mu}) = \widetilde{\sigma}_y \sin(\pi\widetilde{\sigma}_x)\cos(\pi\widetilde{\sigma}_y) & \text{on } \partial\widetilde{\Omega}(\boldsymbol{\mu}), \end{cases} \tag{2.84}$$

where $\widetilde{f}$ is chosen so that the analytical solution is given by $\widetilde{u}(\boldsymbol{x}) = \widetilde{y}\sin(\pi\widetilde{x})\cos(\pi\widetilde{y})$, and the domain is given by the stenosis geometry depictured in the left plot of Figure 2.4, with $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in [-0.5, 0.5] \times [-0.5, 0.5] \times [1, 5]$. As before, the plot on the right in Figure 2.4 we compare the average relative error (in the norm induced by the discrete scalar product $(\cdot,\cdot)_h$) between the FE solution and either its projection onto the reduced space (red line), or the POD-Galerkin approximation (blue line). While up to $L = 20$ both errors show an exponential decay, for greater amounts of POD modes the nonlinear solver applied to the reduced system becomes unstable, leading to an RB error which increases with L. On the other hand, note that the discrepancy between u_h and its projection onto $V_{\mathtt{rb}}$ decreases monotonically within the tested range for L, thus proving the effectiveness of the POD basis

[6] For ease of notation, in the following we omit the subscript $\mathbb{R}^M$ to denote the Euclidean norm on $\mathbb{R}^M$ whenever clear from the contest.

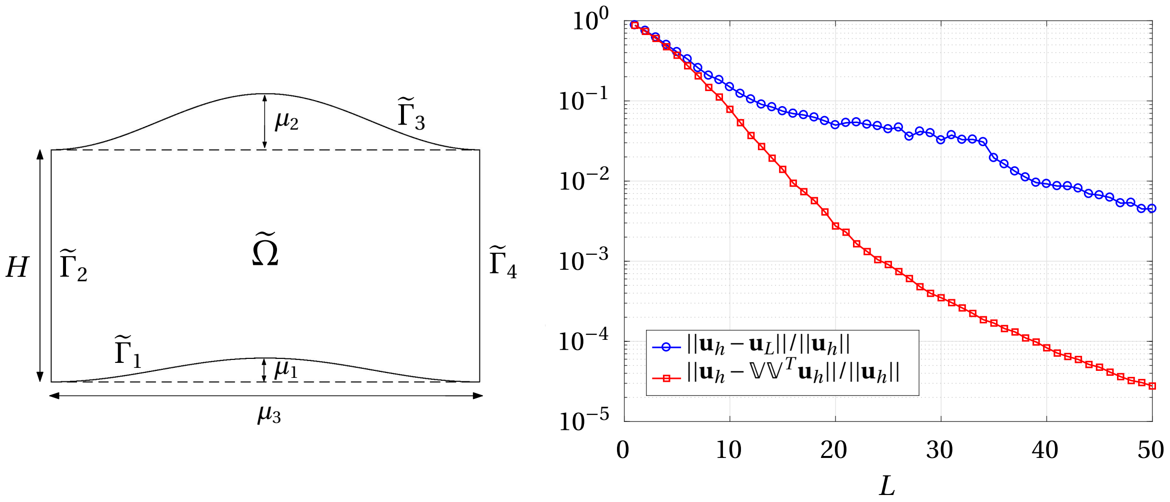

Figure 2.4. Left: computational domain $\widetilde{\Omega} = \widetilde{\Omega}(\boldsymbol{\mu})$ (solid line) for the nonlinear Poisson problem (2.83), with $H = 2$. Right: average relative error committed by approximating the FE solution either through its projection onto the reduced space (red) or the POD-Galerkin RB solution (blue). The errors have been evaluated on a test parameter set $\Xi_{te} \subset \mathscr{P}$ consisting of $N_{te} = 200$ randomly picked values.

functions in catching the principal dynamics characterizing the elements of the (discrete) solution manifold $\mathscr{M}_h$.

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

$$u_h^{\perp}(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{j=1}^{M} \left(\mathbb{V}\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\right)_j \phi_j(\boldsymbol{x}) = \sum_{i=1}^{L} \left(\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\right)_i \, \psi_i(\boldsymbol{x})\,. \tag{2.85}$$

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

$$\begin{aligned} \boldsymbol{\pi} \,:\, \mathscr{P} \subset \mathbb{R}^P &\to \mathbb{R}^L \\ \boldsymbol{\mu} &\mapsto \mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})\,, \end{aligned} \tag{2.86}$$

mapping each input vector parameter $\boldsymbol{\mu} \in \mathscr{P}$ to the coefficients $\mathbb{V}^T\mathbf{u}_h(\boldsymbol{\mu})$ of 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 of the approximation $\hat{\boldsymbol{\pi}}$ of $\boldsymbol{\pi}$, i.e.

$$\mathbf{u}_{\texttt{rb}} = \hat{\boldsymbol{\pi}}(\boldsymbol{\mu})\,, \tag{2.87}$$

and, consequently,

$$\mathbf{u}_L = \mathbb{V}\,\hat{\boldsymbol{\pi}}(\boldsymbol{\mu})\,. \tag{2.88}$$

It worths point out that, provided that the construction of $\hat{\boldsymbol{\pi}}$ is entirely carried out within the offline stage, this approach enables a complete decoupling between the online step and the underlying full-order model, thus ensuring a (potential) dramatic speed up. Moreover, now 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 this fact in the next section.

In the literature, different approaches for the *interpolation* of (2.86) have been developed, e.g., exploiting some geometrical considerations on the solution manifold $\mathscr{M}$ [1], or employing radial basis functions [6]. Whereas, in this work we resort to neural networks, in particular multilayer 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 on 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, exposing the perceptron to a collection of (known) input-output pairs

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

According to the notation and nomenclature introduced 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 $\mathbf{t}_i = \mathbb{V}^T \mathbf{u}_h\big(\boldsymbol{\mu}^{(i)}\big) \in \mathbb{R}^L$ the associated *teaching input*; together, they constitute a *training pattern*. In this respect, note that the teaching inputs $\mathbb{V}^T \mathbf{u}_h\big(\boldsymbol{\mu}^{(i)}\big)$, $i = 1, \dots, N_{tr}$, are generated through the full-order solver. On the one hand, this ensures the reliability of the teaching patterns, given the assumed high-fidelity of the method (conversely to the reduced solver). On the other hand, this also highly suggests to incorporate the learning phase of the preceptron within the offline step of the POD-NN RB method, as described in Algorithm 2.5. 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. Moreover, we have extensively discussed about the time-consuming yet unavoidable *trial-and-error* approach which one should pursue in the research for an optimal network topology. While the Cybenko's theorems (see (i) and (ii) in Section 1.2.2) allow us to confine ourselves to perceptrons with no more than two hidden layers, no similar a priori and general results are available for the number of neurons per layer. Therefore, the speed up enabled by the employment of a neural network-based approach to tackle the *online* queries comes at the cost of an increased *offline* phase.

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) (1.15) 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):

$$
\begin{aligned}
MSE\big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}, \mathbb{V}^T\mathbf{u}_h\big) \propto \big\|\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big\|_{\mathbb{R}^L}^2 &= \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big)^T\big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big) \\
&= \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big)^T \underbrace{\mathbb{V}^T\mathbb{V}}_{\mathbb{I}_L \in \mathbb{R}^{L\times L}} \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}^T\mathbf{u}_h\big) \\
&= \big(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big)^T\big(\mathbb{V}\,\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big) \\
&= \big\|\mathbf{u}_L^{\mathtt{NN}} - \mathbb{V}\,\mathbb{V}^T\mathbf{u}_h\big\|_{\mathbb{R}^M}^2 = \big\|\, u_L^{\mathtt{NN}} - u_h^{\perp}\big\|_h^2 \,,
\end{aligned}
\tag{2.89}
$$

where

$$
u_L^{\mathtt{NN}}(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{i=1}^{L} \big(\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})\big)_i\, \psi_i(\boldsymbol{x}) \in V_{\mathtt{rb}}\,. \tag{2.90}
$$

Therefore, 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.

Algorithm 2.5 The offline and online stages for the POD-NN RB method.

```
function $[\mathbb{V}, \mathscr{N}_{opt}, \mathscr{V}_{opt}, \boldsymbol{w}_{opt}]$ = PODNNOFFLINE($\mathscr{P}, \Omega_h, N, \delta_{\mathtt{POD}}, \delta_{\mathtt{NWT}}, K_{max}, \delta_{\mathtt{NN}}$)
    solve the Laplace problems (2.38) and (2.39), 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 $i = 1,\dots,N$
    $\mathbb{S} = [\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \,|\, \dots \,|\, \mathbf{u}_h(\boldsymbol{\mu}^{(N)})]$
    $\mathbb{V}$ = POD($\mathbb{S}, \delta_{\mathtt{POD}}$)
    design and train a perceptron for the approximation of the map (2.86) up to $\delta_{\mathtt{NN}}$
    $\Rightarrow$ optimal configuration: $\mathscr{N}_{opt}, \mathscr{V}_{opt}, \boldsymbol{w}_{opt}$
end function

function $\mathbf{u}_L^{\mathtt{NN}}(\boldsymbol{\mu})$ = PODNNONLINE($\boldsymbol{\mu}, \mathbb{V}, \mathscr{N}, \mathscr{V}_{opt}, \boldsymbol{w}$)
    evaluate the output $\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}(\boldsymbol{\mu})$ of the network $(\mathscr{N}, \mathscr{V}, \boldsymbol{w})$ 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
```

While we shall numerically prove the actual effectiveness and efficiency of the POD-NN method in 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 a sense of the potentiality of the method.

2.6.1 An a priori error analysis

For the sake of convenience, we directly consider the variational problem (2.21) 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 $u_h^{\perp}(\boldsymbol{\mu}) \in V_{\mathtt{rb}}$ 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 Equation (2.90). 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:

$$\left\| u - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} \leq \| u - u_h \|_{L^2(\Omega)} + \left\| u_h - u_h^{\perp} \right\|_{L^2(\Omega)} + \left\| u_h^{\perp} - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} . \tag{2.91}$$

Let us analyze the three terms appearing on the right-hand side of (2.91). The former quantifies the discrepancy between the exact solution u and the discrete approximation u_h provided by the full-order solver. Throughout the chapter, 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:

$$\| u - u_h \|_{L^2(\Omega)} \leq \| u - u_h \|_V \leq \delta_{\mathtt{HF}} , \tag{2.92}$$

with $\delta_{\mathtt{HF}} > 0$ a given tolerance. Then, the term $\left\| u_h - u_h^{\perp} \right\|_{L^2(\Omega)}$ measures the distance between the truch solution and the reduced space $V_{\mathtt{rb}}$. From the definitions (2.47) and (2.85), it follows that:

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

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:

$$\left\| u_h - u_h^{\perp} \right\|_{L^2(\Omega)}^2 = \left\| \mathbb{M}^{1/2} \left(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\right) \right\|_{\mathbb{R}^M}^2 \leq \| \mathbb{M} \|_2 \left\| \mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h \right\|_{\mathbb{R}^M}^2 , \tag{2.93}$$

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 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 $\mathbb{S} = \left[\mathbf{u}_h\left(\boldsymbol{\mu}^{(1)}\right) \middle| \ldots \middle| \mathbf{u}_h\left(\boldsymbol{\mu}^{(N)}\right)\right] \in \mathbb{R}^{M \times N}$. From (2.78), 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:

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

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 $\mathscr{M}_{\Xi_N}$. Whereas, 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

$$\left\| u_h - u_h^{\perp} \right\|_{L^2(\Omega)}^2 \approx \beta e^{-\alpha L}. \tag{2.94}$$

The last term involved in (2.91) regards the distance between $u_h^{\perp}$ and $u_L^{\mathtt{NN}}$, i.e. the POD-NN solution. Similar to what done for the previous term, recalling the definition (2.90) for $u_L^{\mathtt{NN}}$ we derive:

$$\begin{aligned}
\left\| u_h^{\perp} - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} &= \int_\Omega \left| u_h^{\perp} - u_L^{\mathtt{NN}} \right|^2 d\Omega = \int_\Omega \left| \sum_{i=1}^{M} \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h\right)_i \phi_i - \sum_{i=1}^{M} \left(\mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)_i \phi_i \right|^2 d\Omega \\
&= \sum_{i=1}^{M} \sum_{j=1}^{M} \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)_i \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)_j \underbrace{\int_\Omega \phi_i \phi_j \, d\Omega}_{\mathbb{M}_{i,j}} \\
&= \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right)^T \mathbb{M} \left(\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right) \\
&= \left\| \mathbb{M}^{1/2} \mathbb{V} \left(\mathbb{V}^T \mathbf{u}_h - \mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}}\right) \right\|_{\mathbb{R}^M}^2 \\
&\leq \|\mathbb{M}\|_2 \, \|\mathbb{V}\|_2^2 \left\| \mathbb{V}^T \mathbf{u}_h - \mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} \right\|_{\mathbb{R}^L}^2 \leq L \, \|\mathbb{M}\|_2 \left\| \mathbb{V}^T \mathbf{u}_h - \mathbf{u}_{\mathtt{rb}}^{\mathtt{NN}} \right\|_{\mathbb{R}^L}^2 ,
\end{aligned} \tag{2.95}$$

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 (2.95), by Equation (2.89) the term

$$\left\| \mathbb{V}^T \mathbf{u}_h - \mathbf{u}_L^{\mathtt{NN}} \right\|_{\mathbb{R}^L} \tag{2.96}$$

coincides with the specific error function used to train the network. Then, as for (2.92), (2.96) 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 (2.86) 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 (2.96) 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, plugging (2.92), (2.94) and (2.95) into (2.91), we obtain the following estimate for the POD-NN error in the $L^2(\Omega)$-norm:

$$\left\| u - u_L^{\mathtt{NN}} \right\|_{L^2(\Omega)} \leq \delta_{\mathtt{HF}} + \beta e^{-\alpha L} + \gamma \sqrt{L} \delta_{\mathtt{NN}}, \tag{2.97}$$

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 (2.86).

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] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

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

[9] 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.

[10] 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.

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

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

[13] 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`.

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

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

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

[17] 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.

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

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

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

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

[22] 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.

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

[24] 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.

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

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

[27] 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.

[28] 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.

[29] 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`.

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

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

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

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

[34] 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.

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

[36] 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.

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

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

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

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

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

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

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

[44] Volkwein, S. *Model reduction using proper orthogonal decomposition.* Lecture notes.

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