



**POLITECNICO
DI MILANO**

POLITECNICO DI MILANO
DIPARTIMENTO DI MATEMATICA "F. BRIOSCHI"



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
CHAIR OF COMPUTATIONAL MATHEMATICS AND SIMULATION SCIENCE

Reduced order modeling of nonlinear problems using neural networks

Master of Science in
Computational Science and Engineering

Candidate:

Stefano Ubbiali
ID Number 836697

Supervisors:

Prof. Jan S. Hesthaven
Prof. Matteo Zunino

Academic Year 2016 - 2017

Contents

1	Introduction to neural networks	1
1.1	Biological motivation	1
1.2	Artificial neural networks	3
1.2.1	Neuronal model	4
1.2.2	Network topologies: the feedforward neural network	7
1.2.3	Training a multilayer feedforward neural network	9
1.2.4	Practical considerations on the design of artificial neural networks . .	20
2	Reduced basis methods for nonlinear partial differential equations	23
2.1	Parametrized nonlinear PDEs	27
2.1.1	Nonlinear Poisson equation	27
2.1.2	Steady Navier-Stokes equations	29
2.2	From the original to the reference domain	30
2.2.1	Change of variables formulae	32
2.2.2	The problems of interest	34
2.2.3	The boundary displacement-dependent transfinite map (BDD TM) . .	35
2.3	Well-posedness of the test cases	37
2.4	Finite element method	38
2.4.1	Nonlinear Poisson equation	41
2.4.2	Steady Navier-Stokes equations	42
2.5	POD-Galerkin reduced basis method	44
2.5.1	Proper Orthogonal Decomposition	47
2.5.2	Implementation: details and issues	53
2.5.3	Application to the steady Navier-Stokes equations	54
2.6	A POD-based RB method using neural networks	56
2.6.1	An a priori error analysis	63
3	Numerical results	67
3.1	One-dimensional Poisson equation	69
3.1.1	Linear test case	69
3.1.2	Nonlinear test case, two parameters	71
3.1.3	Nonlinear test case, three parameters	73
3.2	Two-dimensional Poisson equation	75
3.2.1	Linear test case	77
3.2.2	Nonlinear test case, two parameters	79
3.2.3	Nonlinear test case, three parameters	80

List of Figures

1.1	Simplified representation of a biological neuron, with the components discussed in the text; retrieved from [28].	2
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.	4
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.	6
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.	7
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.	10
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	36
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.	37
2.3	The parametrized computational domain (solid line) for the Poisson problem (2.89).	58
2.4	Left: average relative error between the FE solution for the problem (2.89) and either its projection onto the reduced space (red) or the POD-Galerkin RB solution (blue); the errors have been evaluated on a test parameter set $\Xi_{te} \subset \mathcal{P}$ consisting of $N_{te} = 50$ randomly picked values. Right: comparison between the online run times for the FE (full-order) scheme and the POD-Galerkin (reduced-order) method on Ξ_{te}	59
2.5	Three point sets randomly generate through the latin hypercube sampling. Observe the good coverage provided by the union of the sets, with only a few overlapping points.	61

3.1	Left: error analysis for the POD-G RB method applied to Equation (3.4); the results refer to a POD basis constructed based on $N = 25, 50, 100$ snapshots; for $N = 100$, the singular values of the correspondent snapshot matrix are shown as well. Right: FE and POD-G solutions for three parameter values. . .	69
3.2	In respect to the approximation of the map (2.91) for the Poisson problem (3.4) via neural networks, comparison between three supervised algorithms - Levenberg-Marquardt (blue), Scaled Conjugate Gradient (orange), BFGS Quasi-Newton (yellow) - in terms of the optimal configuration detected (<i>right</i>) and the associated test error (<i>right</i>) for different amounts of training samples.	70
3.3	Left: relative error yielded by the POD-NN RB method applied to Equation (3.4); the convergence to the projection error for $L = 10$ (solid line) is analyzed in terms of both the number of neurons included in the neural network and the dimension of the training set. Right: FE and POD-NN solutions for three parameter values.	70
3.4	Error analysis for the POD-Galerkin RB method applied to the problem (3.5). The lower-bound provided by the projection error (3.3) is reported as reference.	71
3.5	Left: error analysis for the POD-NN RB method (dotted lines) applied to problem (3.5). The solid solid sections refer to the steps carried out by the routine 2.5; the error yielded by the POD-G method using $L = 8$ or $L = 15$ basis functions are reported as reference. Right: comparison between FE and POD-NN solutions for three input vectors.	72
3.6	From left to right, top to bottom: regression plots for the first, seventh, eighth and nine scalar components of the outputs provided on Ξ_{te} by a network with $H_1 = H_2 = 25$ neurons per hidden layer, trained to approximate the map (2.91) for problem (3.5).	73
3.7	The first 10 POD basis functions for the (nonlinear) Poisson problem (3.6). .	74
3.8	Convergence analysis for the POD-G and POD-NN methods applied to the problem (3.6) (<i>left</i>) and comparison between the FE and the POD-NN solutions for three parameter values (<i>right</i>). These latter results have been obtained via a neural network with $H_1 = H_2 = 35$ units per hidden layer, and considering $L = 10$ POD modes.	74
3.9	Sensitivity analysis for the POD-NN method applied to the problem (3.6) with respect to the number of POD coefficients (per sample) used during the training; to prevent overfitting, $N_{tr} = 300$ learning patterns have been used. .	75
3.10	The quadrilateral domain used in the simulations (<i>left</i> , solid line) and the parametrizations of its sides (<i>right</i>).	76
3.11	The stenosis geometry employed in the simulations (<i>left</i>) and the parametrizations of its sides (<i>right</i>).	76
3.12	Online run times for the FE, POD-G and POD-NN methods applied to the problem (3.8), for $N_{te} = 100$ randomly generated parameter values.	78

3.13	On the right, error analysis for the POD-G (blue dashed line) and the POD-NN (dotted lines) methods applied to the problem (3.8). For the latter, a convergence analysis with respect to the number of training samples and hidden neurons used is provided on the left. The solid tracts denote the steps actually pursued by the automatic training routine 2.5. All the results refer to neural networks exposed to $L = 15$ POD coefficients per learning sample.	78
3.14	Finite element (<i>left</i>), POD-G (<i>center</i>) and POD-NN (<i>right</i>) solution to the semilinear Poisson problem (3.12) re-stated over the reference domain, with $\boldsymbol{\mu} = (0.835, 0.034)$	79
3.15	Online relative errors (<i>left</i>) and run times (<i>right</i>) for the POD-G and the POD-NN methods applied to problem (3.12) for $N_{te} = 50$ randomly picked parameter values.	80
3.16	Convergence analysis for the POD-NN RB method applied to the BVP (3.12). The results have been obtained via three-layers (i.e., two hidden layers plus the output layer) neural networks with $H_1 = H_2 \in \{15, 20, 25, 30, 35\}$ neurons in each hidden layer, and trained with $N_{tr} \in \{100, 200, 300\}$ learning patterns. Each pattern consists of an input vector $\boldsymbol{\mu} \in \mathcal{P}$ and the first $L = 35$ POD coefficients $(\mathbf{u}_h(\boldsymbol{\mu}), \boldsymbol{\psi}_i)_{\mathbb{R}^M}$, $i = 1, \dots, L$, as teaching inputs.	80
3.17	On the left, error analysis for the POD-G (blue dashed line) and the POD-NN (dotted lines) methods (<i>left</i>) applied to the problem (3.13). For the latter, a convergence analysis with respect to the number of training samples and hidden neurons used is provided on the right. The solid tracts denote the steps actually pursued by the automatic training routine 2.5. All the results refer to neural networks exposed to $L = 30$ POD coefficients per sample during the learning step.	81
3.18	82

List of Algorithms

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.	13
1.2	Backbone of any supervised offline learning algorithm; the procedure to compute the accumulated error is provided as well.	14
1.3	An iteration of the Levenberg-Marquardt training algorithm.	19
1.4	The complete training algorithm adopted in our numerical tests.	22
2.1	The Newton's method applied to the nonlinear system (2.49).	41
2.2	The Newton's method applied to the reduced nonlinear system (2.67).	48
2.3	The POD algorithm.	52
2.4	The offline and online stages for the POD-Galerkin (POD-G) RB method.	53
2.5	Selection of an optimal network configuration.	61
2.6	The offline and online stages for the POD-NN RB method.	62

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., [15, 17, 28], 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 3, 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 [17].

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

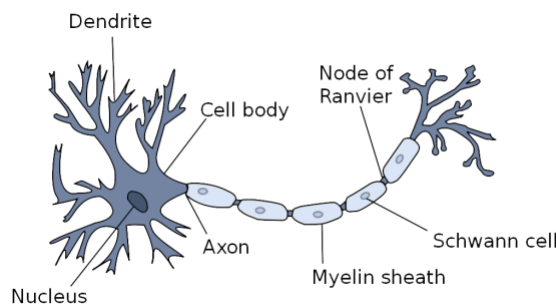


Figure 1.1. Simplified representation of a biological neuron, with the components discussed in the text; retrieved from [28].

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 [28].

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 constituting an extension of the soma and splitting in many branches at the opposite extremity [46]. 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 transferred 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 postsynaptic 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

¹For 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 [15]. That is, the set of interneuron connection strengths represent the information storage, i.e., the knowledge, of a neural network [28]. 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 possibility to derive a set of tools which will let it provide reasonable solutions in similar circumstances. In other terms, the cognitive system should be able to *generalize*. Furthermore, after a successful training a neural network should 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 [28].

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 [36]. 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 [46].

Formally, a neural network could be defined as follows [28].

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.

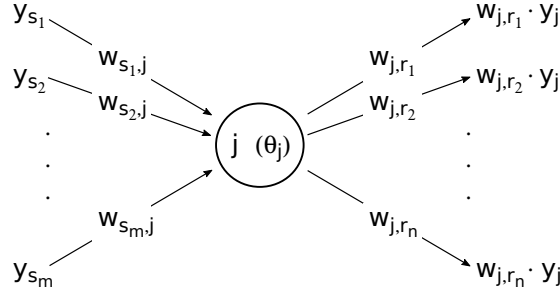


Figure 1.2. Visualization of the generic j -th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j} y_{s_1}, \dots, w_{s_m,j} y_{s_m}\}$ coming from the sending neurons s_1, \dots, s_m , and fires y_j , sent to the target neurons $\{r_1, \dots, r_n\}$ through 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 $\mathbf{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ into a scalar u_j often called *net input*, i.e.,

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

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:

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

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 $\theta_j \in \mathbb{R}$ [28]:

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

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 [28].

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

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

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

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

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:

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

Neurons characterized by such an activation function are usually named McCulloch-Pitts neurons, after the seminal work of McCulloch and Pitts [17], 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 [17]. Sigmoid functions are s-shaped and monotonically increasing, and assume values in a bounded interval, typically $[0, 1]$, as the logistic function,

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

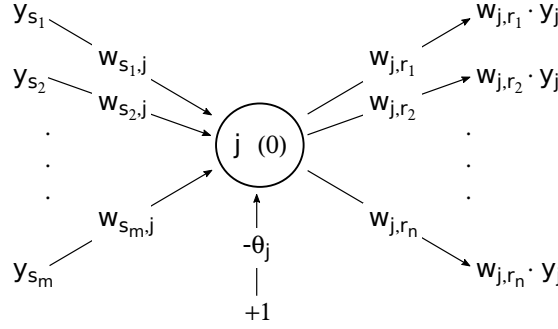


Figure 1.3. Visualization of the generic j -th neuron of an artificial neural network. The neuron accumulates the weighted inputs $\{w_{s_1,j} y_{s_1}, \dots, w_{s_m,j} y_{s_m}, -\theta_j\}$ coming from the sending neurons s_1, \dots, s_m, b , respectively, with b the bias neuron. The neuron output y_j 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,

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

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:

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

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 $\mathbf{p} = [y_{s_1}, \dots, y_{s_m}]^T \in \mathbb{R}^m$ of the neuron is generally vectorial, i.e., $m > 1$, the output is scalar. The output y_j could then 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) [17]. 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.

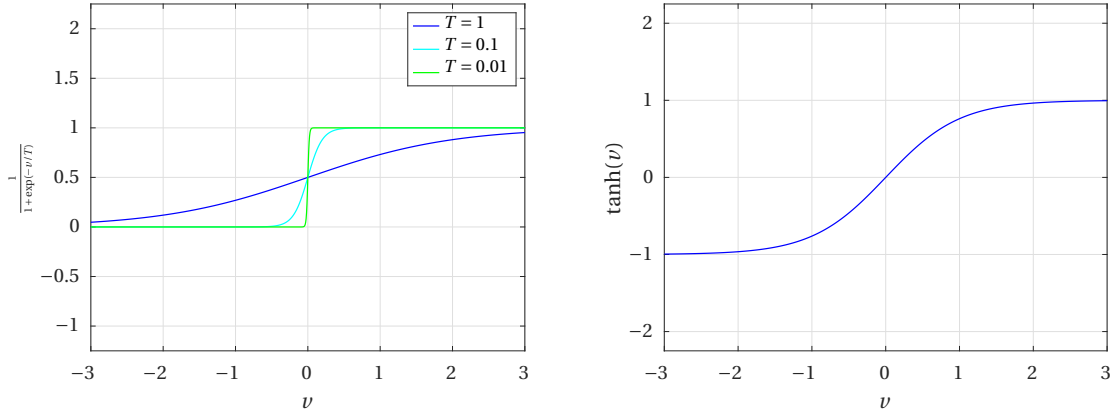


Figure 1.4. Left: logistic function (1.7) for three values of the parameter T ; note that as T decreases, the logistic function resembles the Heaviside function. Right: hyperbolic tangent.

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* [44], neurons are arranged into *layers*, with one *input layer* of M_I source neurons, K *hidden layers*, each one consisting of H_k computing neurons, $k = 1, \dots, K$, and an *output layer* of M_O computing neurons. As characteristic property, neurons in a layer can only be connected with neurons in the next layer towards the output layer. Then, an *activation pattern* $\mathbf{p} \in \mathbb{R}^{M_I}$, supplied to the network through the source nodes in the first layer, provides the input signal for the neurons in the first hidden layer. For each hidden layer, its output signal gives the input pattern for the following layer. In this way, information travels towards the last layer of the network, i.e., the output layer, whose outputs constitute the components of the overall output $\mathbf{q} \in \mathbb{R}^{M_O}$ of the network, response to the input pattern \mathbf{p} ². Hence, a feedforward network establishes a map between the *input space* \mathbb{R}^{M_I} and the *output space* \mathbb{R}^{M_O} . This makes this network architecture particularly suitable for, e.g., classification and continuous function approximation.

Feedforward networks can be classified according to the number of hidden 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 provided with a unique output neuron, equipped with a binary

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

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 [28]. 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 [8, 9]. In detail:

- (i) a multi-layer perceptron with one layer of *hidden neurons* and differentiable activation functions can approximate any *continuous* function [9];
- (ii) a multi-layer perceptron with two layers of *hidden neurons* and differentiable activation functions can approximate *any function* [8].

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 1.5. In this case, we have $M_I = 3$ input neurons (denoted with the letter i), $H_1 = H_2 = 6$ hidden neurons (letter h), 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 [22], inspired by the behaviour of electrically charged particles within a magnetic field, and the self-organizing maps by Kohonen [27], 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 ϕ between the parameters μ of a parametrized partial differential equation, and the coefficients α 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 2, 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 multi-layer 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 [17]. Specifically, this entails that after a successful 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* [28].

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

Unsupervised learning. Although supervised learning is a simple and intuitive paradigm, there exist many tasks which require a different approach 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 [28]. 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

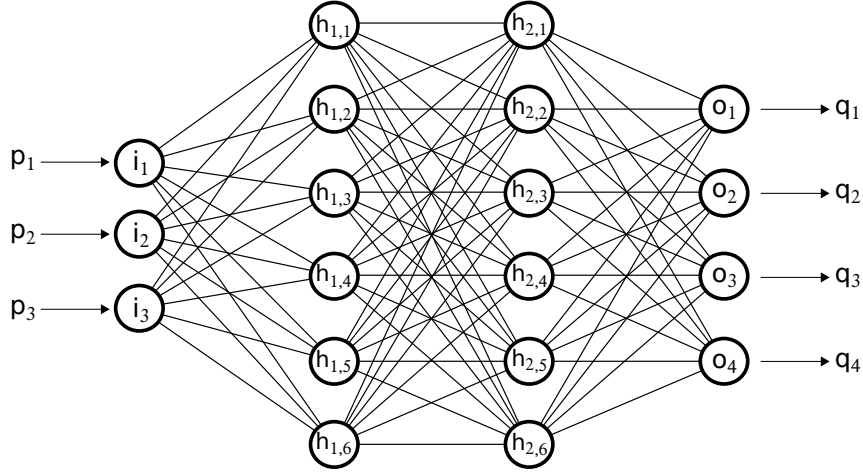


Figure 1.5. A three-layer feedforward neural network, with 3 input neurons, two hidden layers each one consisting of 6 neurons, and 4 output neurons. Within each connection, information flows from left to right.

between a long-term and a short-term reward [25].

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 f ,

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

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

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

where $F : \mathbb{R}^{M_I} \rightarrow \mathbb{R}^{M_O}$ 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 f for each input in the domain V , the system should provide accurate predictions for the input patterns, i.e.,

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

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

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

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 [18]. Inspired by neurobiological considerations, the rule can be stated in a two-steps fashion [17]:

- (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 [17]. 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

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

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

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

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) [28]:

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

with the functions h and g dependent on the specific learning rule, and t_j the j -th component of the teaching input \mathbf{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

$$E = E(\mathbf{w}) \in \mathbb{R}. \quad (1.13)$$

Recalling the notation and assumptions introduced in Definition 1.1, here $\mathbf{w} \in \mathbb{R}^{|\mathcal{V}|}$ 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 network³. 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 \mathbf{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* $E_{\mathbf{p}}(\mathbf{w})$, specific to the activation pattern \mathbf{p} , and the *total error* $E(\mathbf{w})$ accounting for all specific errors, namely

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

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

$$E_{\mathbf{p}}(\mathbf{w}) = \frac{1}{M_O} \sum_{j=1}^{M_O} (t_{\mathbf{p},j} - q_{\mathbf{p},j})^2, \quad (1.15)$$

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

$$E(\mathbf{w}) = \sum_{\mathbf{p} \in P} E_{\mathbf{p}}(\mathbf{w}) = \sum_{\mathbf{p} \in P} \frac{1}{M_O} \sum_{j=1}^{M_O} (t_{\mathbf{p},j} - q_{\mathbf{p},j})^2. \quad (1.16)$$

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 $E(\mathbf{w})$. Therefore, to increase the performance of the network, we need to iteratively move toward a (local) minimum of the surface [17]. For this purpose, we may employ a *steepest descent* technique, thus following the direction of the anti-gradient, i.e.,

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

$\eta > 0$ 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* [35] 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 function⁴. The derivation of the backpropagation algorithm is provided in the following.

³Please 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 \mathbf{w} - which is a practical advantage.

⁴Therefore, 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 $(\mathcal{N}, \mathcal{V}, \mathbf{w}_0)$, training set $P = \{\mathbf{p}_i, \mathbf{t}_i\}_{i=1}^{N_{tr}}$,
metric $d(\cdot, \cdot) : \mathbb{R}^{M_o} \times \mathbb{R}^{M_o} \rightarrow \mathbb{R}$, tolerance ϵ , maximum number of epochs T

Output: trained neural network $(\mathcal{N}, \mathcal{V}, \mathbf{w}_{opt})$

```

1:  $t = 0, i = 1, k = 0$ 
2:  $\mathbf{w}(0) = \mathbf{w}_0$ 
3: while  $t < T$  and  $k < N$  do
4:   evaluate output vector  $\mathbf{y}_{\mathbf{p}_i}(t)$ , correspondent to input pattern  $\mathbf{p}_i$ 
5:    $E_{\mathbf{p}_i}(\mathbf{w}(t)) = d(\mathbf{y}_{\mathbf{p}_i}(t), \mathbf{t}_1)$ 
6:   if  $E_{\mathbf{p}_i}(\mathbf{w}(t)) < \epsilon$  then
7:      $k \leftarrow k + 1$ 
8:   else
9:      $k = 0$ 
10:    compute weight update  $\Delta \mathbf{w}(t)$  based on  $E_{\mathbf{p}_i}(\mathbf{w}(t))$ 
11:     $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$ 
12:   end if
13:    $t \leftarrow t + 1, i \leftarrow i \pmod{N} + 1$ 
14: end while
15:  $\mathbf{w}_{opt} = \mathbf{w}(t)$ 

```

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 $E(\mathbf{w})$ in an arbitrary point \mathbf{w} , thus allowing to compute the update $\Delta w_{i,j}$ for the weight $w_{i,j}$ according to (1.17). 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 \mathbf{p} , thus to explicitly specify the input pattern \mathbf{p} we refer to.

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

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

Let us focus on the product

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

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 $(\mathcal{N}, \mathcal{V}, \mathbf{w}_0)$, training set $P = \{\mathbf{p}_i, \mathbf{t}_i\}_{i=1}^{N_{tr}}$,
metric $d(\cdot, \cdot) : \mathbb{R}^{M_o} \times \mathbb{R}^{M_o} \rightarrow \mathbb{R}$, tolerance ϵ , maximum number of epochs T

Output: trained neural network $(\mathcal{N}, \mathcal{V}, \mathbf{w}_{opt})$

```

1:  $t = 0$ 
2:  $\mathbf{w}(0) = \mathbf{w}_0$ 
3:  $E(\mathbf{w}(0)) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w}(0), P, d)$ 
4: while  $t < T$  and  $E(\mathbf{w}(t)) > \epsilon$  do
5:   compute weight update  $\Delta \mathbf{w}(t)$  based on  $E(\mathbf{w}(t))$ 
6:    $\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$ 
7:    $E(\mathbf{w}(t+1)) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w}(t+1), P, d)$ 
8:    $t \leftarrow t+1$ 
9: end while
10:  $\mathbf{w}_{opt} = \mathbf{w}(t)$ 
11: function  $E(\mathbf{w}) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w}, P, d)$ 
12:    $E(\mathbf{w}) = 0$ 
13:   for  $i = 1, \dots, N_{tr}$  do
14:     evaluate output vector  $\mathbf{y}_{\mathbf{p}_i}$ , correspondent to input pattern  $\mathbf{p}_i$ 
15:      $E(\mathbf{w}) \leftarrow E(\mathbf{w}) + d(\mathbf{y}_{\mathbf{p}_i}, \mathbf{t}_i)$ 
16:   end for
17: end function

```

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

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

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

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

$\delta_{\mathbf{p},j}$ is often referred to as the *sensitivity* of the specific error $E_{\mathbf{p}}$ 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:

$$\Delta w_{i,j} = \eta \sum_{\mathbf{p} \in P} \delta_{\mathbf{p},j} y_{\mathbf{p},i}. \quad (1.22)$$

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

follows:

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

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 [28]. Hence:

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

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):

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

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:

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

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

$$\delta_{\mathbf{p},j} = \begin{cases} f'_{act}(u_{\mathbf{p},j}) \sum_{r \in R} \delta_{\mathbf{p},r} w_{j,r}, & \text{if } j \text{ inner neuron} , \\ \frac{2}{M_O} f'_{act}(u_{\mathbf{p},j}) (t_{\mathbf{p},j} - y_{\mathbf{p},j}), & \text{if } j \text{ output neuron} . \end{cases} \quad (1.26a)$$

Some relevant remarks about the overall algorithm should be pointed out. First, observe that Equation (1.26a) defines $\delta_{\mathbf{p},j}$ for a hidden node j by relying on neurons in the following layer, whereas Equation (1.26b) only involves variables concerning the neuron (namely, $u_{\mathbf{p},j}$ and $y_{\mathbf{p},j}$) and the exact output $t_{\mathbf{p},j}$. 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 [28, 48].

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 readily 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 [28], 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 1.4, 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 [28]. Hence, different alternatives to backpropagation have been proposed in literature, either by modifying the original algorithm (as, e.g., the resilient backpropagation [43] or the quickpropagation [12]), 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 [30] is an approximation to the Newton's method [14]. As for backpropagation, the learning

procedure is driven by the loss function $E = E(\mathbf{w})$, $\mathbf{w} \in |\mathcal{V}|$. Applying the Newton's method for the minimization of E , at each iteration the *search direction* $\Delta \mathbf{w}$ is found by solving the following linear system:

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

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

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

with $e_{\mathbf{p},j}$ the j -th component of the error vector $\mathbf{e}_{\mathbf{p}} = \mathbf{t}_{\mathbf{p}} - \mathbf{y}_{\mathbf{p}}$ corresponding to the input pattern \mathbf{p} . Then, introducing the Jacobian $J_{\mathbf{p}}$ of the specific error vector $\mathbf{e}_{\mathbf{p}}$ with respect to \mathbf{w} , i.e.,

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

simple computations yield:

$$\nabla E(\mathbf{w}) = \sum_{\mathbf{p} \in P} \frac{2}{M_O} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}} \in \mathbb{R}^{|\mathcal{V}|} \quad (1.30)$$

and

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

with

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

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

$$\left[\sum_{\mathbf{p} \in P} J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w}) \right] \Delta \mathbf{w} = - \sum_{\mathbf{p} \in P} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}}(\mathbf{w}). \quad (1.32)$$

The Levenberg-Marquardt modification to the Newton's method reads [14, 30]:

$$\left[\sum_{\mathbf{p} \in P} J_{\mathbf{p}}(\mathbf{w})^T J_{\mathbf{p}}(\mathbf{w}) + \mu I \right] \Delta \mathbf{w} = - \sum_{\mathbf{p} \in P} J_{\mathbf{p}}(\mathbf{w})^T \mathbf{e}_{\mathbf{p}}(\mathbf{w}), \quad (1.33)$$

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 $\Delta \mathbf{w}$ 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 $\Delta \mathbf{w}(t)$ leads to a reduction in the performance function, i.e., $E(\mathbf{w}(t) + \Delta \mathbf{w}(t)) < E(\mathbf{w}(t))$, the parameter μ is reduced by a factor $\beta > 1$. Conversely, if $E(\mathbf{w}(t) + \Delta \mathbf{w}(t)) > E(\mathbf{w}(t))$ the parameter is multiplied by the same factor β . This reflects the idea that far from the actual minimum we should prefer the gradient method to 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 [30].

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

$$\frac{\partial e_{\mathbf{p},h}}{\partial w_k} = \frac{\partial e_{\mathbf{p},h}}{\partial w_{i,j}}, \quad 1 \leq h \leq M_O, 1 \leq k \leq |\mathcal{V}|. \quad (1.34)$$

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:

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

with

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

For the computation of the derivative $\partial e_{\mathbf{p},h} / \partial y_{\mathbf{p},j}$ 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.,

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

We can then distinguish three cases:

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

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} = -1; \quad (1.37)$$

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

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} = 0; \quad (1.38)$$

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

$$\frac{\partial e_{\mathbf{p},h}}{\partial y_{\mathbf{p},j}} = \sum_{r \in R} \frac{\partial e_{\mathbf{p},h}}{\partial u_{\mathbf{p},r}} \frac{\partial u_{\mathbf{p},r}}{\partial y_{\mathbf{p},j}} = - \sum_{r \in R} \delta_{\mathbf{p},h,r} w_{j,r}. \quad (1.39)$$

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

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

with

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

$$(1.40b)$$

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

Let us finally remark that a trial and error approach is still required to find satisfactory values for μ and β ; as proposed in [30], 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 [14]. However, it is more efficient than backpropagation for networks with a few hundreds of connections, besides producing much more accurate results. We shall gain further insights into this topic in Chapter 3.

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

```

1: function  $[\mathbf{w} + \Delta \mathbf{w}, E(\mathbf{w} + \Delta \mathbf{w}), \mu] = \text{LMITERATION}(\mathcal{N}, \mathcal{V}, \mathbf{w}, P, E(\mathbf{w}), d, \mu)$ 
2:    $\beta = 10$ 
3:   for  $i = 1, \dots, N_{tr}$  do
4:     evaluate output vector  $\mathbf{y}_{\mathbf{p}_i}$ , correspondent to input pattern  $\mathbf{p}_i$ 
5:      $\mathbf{e}_{\mathbf{p}_i} = \mathbf{y}_{\mathbf{p}_i} - \mathbf{t}_{\mathbf{p}_i}$ 
6:     for  $h = 1, \dots, M_0, k = 1, \dots, |\mathcal{V}|$  do
7:       compute  $(J_{\mathbf{p}_i})_{h,k}$  according to (1.35), (1.40a) and (1.40b)
8:     end for
9:   end for
10:  assemble and solve  $\left[ \sum_{i=1}^{N_{tr}} J_{\mathbf{p}_i}^T J_{\mathbf{p}_i} + \mu I \right] \Delta \mathbf{w} = - \sum_{i=1}^{N_{tr}} J_{\mathbf{p}_i}^T \mathbf{e}_{\mathbf{p}_i}$ 
11:   $E(\mathbf{w} + \Delta \mathbf{w}) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w} + \Delta \mathbf{w}, P_{tr}, d)$ 
12:  if  $E(\mathbf{w} + \Delta \mathbf{w}) < E(\mathbf{w})$  then
13:     $\mu \leftarrow \mu / \beta$ 
14:  else
15:     $\mu \leftarrow \mu * \beta$ 
16:  end if
17: 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 empirical 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* [28]. 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 parameters, 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 $L(\mu)$, namely

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

Here, L is a functional increasing with the components of \mathbf{w} , thus penalizing large values for the weights, and so preventing the network from heavily relying on individual data points [31]. 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 [26]. 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 $\Delta \mathbf{w}$. 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 [31]. For further details on the way such subsets have been generated for our test cases, we refer the reader to the upcoming chapters.

At the beginning of the learning stage, the error on both the training and 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 \mathbf{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 [20]) to iteratively remove the *less relevant* connection, erasing a neuron whenever it gets isolated from the rest of the network.

As we shall see in Section 2.6, in this work we resort to the first approach for the sake of implementation-wise convenience. In particular, we first consider perceptrons with a single hidden layer, and we then add another computing layer whenever necessary⁵

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 \mathbf{w}_0 , we train each network topology several times, say K_{mr} , employing different \mathbf{w}_0^k , $k = 1, \dots, K_{mr}$, finally keeping the configuration yielding the minimum error on the test data set. This approach is usually referred to as *multiple-restarting*. The definitive training procedure used in this project, based on the Levenberg-Marquardt algorithm and keeping into consideration all the observations made in this section, is given in Algorithm 1.4.

⁵Recall 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.

```

1: function  $[\mathbf{w}_{opt}, E_{opt}] = \text{TRAINING}(\mathcal{N}, \mathcal{V}, P_{tr}, P_{va}, P_{te}, d, K_{ms}, \epsilon, T, K_{ea})$ 
2:    $E_{opt} = \infty$ 
3:   for  $j = 1, \dots, K_{ms}$  do
4:      $t = 0, k = 0, \mu = 0.01$ 
5:     randomly generate  $\mathbf{w}(0)$ 
6:      $E_{tr}(\mathbf{w}(0)) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w}(0), P_{tr}, d)$ 
7:     while  $t < T$  and  $E_{tr}(\mathbf{w}(t)) > \epsilon$  and  $k < K_{ea}$  do
8:        $[\mathbf{w}(t+1), E_{tr}(\mathbf{w}(t+1)), \mu] = \text{LMITERATION}(\mathcal{N}, \mathcal{V}, \mathbf{w}(t), P_{tr}, E(\mathbf{w}(t)), d, \mu)$ 
9:        $E_{va}(\mathbf{w}(t+1)) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w}(t+1), P_{va}, d)$ 
10:      if  $E_{va}(\mathbf{w}(t+1)) > E_{va}(\mathbf{w}(t))$  then
11:         $k \leftarrow k + 1$ 
12:      else
13:         $k = 0$ 
14:      end if
15:       $t \leftarrow t + 1$ 
16:    end while
17:     $\mathbf{w}_{opt}^{(j)} = \mathbf{w}(t - k)$ 
18:     $E_{te}(\mathbf{w}_{opt}^{(j)}) = \text{OFFLINEERROR}(\mathcal{N}, \mathcal{V}, \mathbf{w}_{opt}^{(j)}, P_{te}, d)$ 
19:    if  $E_{te}(\mathbf{w}_{opt}^{(j)}) < E_{opt}$  then
20:       $\mathbf{w}_{opt} = \mathbf{w}_{opt}^{(j)}$ 
21:       $E_{opt} = E_{te}(\mathbf{w}_{opt}^{(j)})$ 
22:    end if
23:  end for
24: 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 incompressible Navier-Stokes equations are used as expository vehicles. The latter feature a quadratic nonlinearity laying in the convective term, while we consider instances of the Poisson equation where the nonlinearity stems 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 almost effortlessly extended to higher dimensions, and to other classes of either linear or nonlinear PDEs as well.

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

Let $\boldsymbol{\mu}_{ph} \in \mathcal{P}_{ph} \subset \mathbb{R}^{P_{ph}}$ and $\boldsymbol{\mu}_g \in \mathcal{P}_g \subset \mathbb{R}^{P_g}$ 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 $\tilde{\Omega} = \tilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 1, 2$. Furthermore, assume both \mathcal{P}_{ph} and \mathcal{P}_g compact (i.e., closed and bounded) subsets of $\mathbb{R}^{P_{ph}}$ and \mathbb{R}^{P_g} , respectively, and denote by $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathcal{P} = \mathcal{P}_{ph} \times \mathcal{P}_g \subset \mathbb{R}^P$, $P = P_{ph} + P_g$, the overall *input vector parameter*. For a given $\boldsymbol{\mu}$ in the parameter space \mathcal{P} , we then seek the corresponding solution $\tilde{u} = \tilde{u}(\boldsymbol{\mu})$ of the underlying PDE in a suitable Hilbert space $\tilde{V}(\boldsymbol{\mu}_g)$, defined over the computational domain $\tilde{\Omega}(\boldsymbol{\mu}_g)$.

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

$$\Phi : \Omega \times \mathcal{P}_g \rightarrow \tilde{\Omega},$$

enabling the formulation and resolution of the differential problem over a fixed, parameter-independent, reference domain Ω [33], such that

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

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

$$u : \mathcal{P} \rightarrow V. \quad (2.1)$$

The map (2.1) defines the *solution manifold* $\mathcal{M} = \{u = u(\boldsymbol{\mu}) : \boldsymbol{\mu} \in \mathcal{P}\} \subset V$, consisting of solutions to the parametrized PDE for any admissible input parameter [21]. 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 $\Phi(\boldsymbol{\mu}_g)$ given the boundary parametrization of $\tilde{\Omega}(\boldsymbol{\mu})$ [24].

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 schemes, 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:

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

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

Many applications entail the repeated resolution of a parametrized PDE in various settings and over different geometrical configurations, namely, for several parameter values. Notable examples include parametric sensitivity analysis, optimal control, topology optimization, and uncertainty quantification. From a geometrical standpoint, one has then to span the *discrete solution manifold* $\mathcal{M}_h = \{u_h = u_h(\boldsymbol{\mu}) : \boldsymbol{\mu} \in \mathcal{P}\} \subset V_h$. 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, and even on massive parallel workstations [41].

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 variegated 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* $V_{rb} \subset V_h$, approximating the discrete solution manifold \mathcal{M}_h by means of L well-chosen *basis functions* $\{\psi_1, \dots, \psi_L\} \subset V_h$, namely

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

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

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

so that

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

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

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

The reduced system (2.5) typically 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 $\text{Col}(\mathbb{V})$ of \mathbb{V}^1 [33]. 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 [39]. 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 [21]. 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 the cost of any online query is uniquely related to the dimension L of the reduced space V_{rb} [41]. Whereas, for differential problems showing a non-affine parametric dependence, as the ones considered in this work, one needs some further machinery to guarantee a significant computational saving with respect to the underlying full-order scheme. In this respect, one may recover an affine approximation of the differential operator [33]; to this aim, the discrete empirical interpolation method (DEIM) (see, e.g., [3]) and its matrix version (MDEIM) (see, e.g., [37]) represent the state-of-the-art nowadays. Although their established effectiveness, we draw the attention to the fact that these techniques unavoidably introduce a new source of inaccuracy in the model, and as a result one may need to retain a larger number of basis functions to guarantee a given degree of precision.

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 exact solution u , play an important role in the *certification* of a reduced order method, thus enabling the user to trust the output provided by the method itself [21]. 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 stability and convergence theories to nonlinear, non-affine problems, e.g., the Navier-Stokes equations [41], relying on advanced functional tools. In particular, the numerical stability at the reduced level may not naturally follow from the stability of the underlying high-fidelity scheme, thus representing

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

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

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

a challenge for any RB technique applied to nonlinear problems [2]; for the Navier-Stokes equations, further details on this point are provided in Section 2.5.3. Moreover, a complete error analysis should also 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). Indeed, iterative methods may provide a quite slow convergence to the root. As we shall further investigate in Section 2.6, this could be ascribed to the dense nature of the Jacobian of the system.

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

$$\mathbf{u}_h^\mathbb{V}(\boldsymbol{\mu}) = \underset{\mathbf{w} \in \text{Col}(\mathbb{V})}{\text{argmin}} \|\mathbf{w} - \mathbf{u}_h(\boldsymbol{\mu})\| = \mathbb{V} \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) = \sum_{l=1}^L (\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}))_l \boldsymbol{\psi}_l. \quad (2.6)$$

In particular, we note that

$$\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}) \quad (2.7)$$

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

$$\begin{aligned} \boldsymbol{\pi} : \mathcal{P} \subset \mathbb{R}^P &\rightarrow \mathbb{R}^L \\ \boldsymbol{\mu} &\mapsto \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}). \end{aligned} \quad (2.8)$$

The approximation might be accomplished either through *interpolation* [1, 7] or *regression* [16] of a collection of target input-output pairs $\{\boldsymbol{\mu}^{(i)}, \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^{N_{tr}}$. Here, we pursue the latter approach, employing multi-layer feedforward neural networks, presented in Chapter 1. In this way, for any $\boldsymbol{\mu} \in \mathcal{P}$ the reduced-order approximation to the $\boldsymbol{\mu}$ -parametrized PDE is obtained as

$$\mathbf{u}_L^{\text{NN}}(\boldsymbol{\mu}) = \mathbb{V} \hat{\boldsymbol{\mu}}^{\text{NN}}(\boldsymbol{\mu}) \in \text{Col}(\mathbb{V}) \subset \mathbb{R}^M \quad (2.9)$$

with $\hat{\boldsymbol{\mu}}^{\text{NN}}$ the input-output map established by the neural network. In other terms, the online phase reduces to the (fast) evaluation of the map $\hat{\boldsymbol{\mu}}^{\text{NN}}$, without involving further algebraic operations. 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 thoroughly 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 incompressible Navier-Stokes equations). In doing so, we aim at showing how the proposed methodology could be almost effortlessly extended to other differential problems.

2.1 Parametrized nonlinear PDEs

Recalling the input vector parameter $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathcal{P} = \mathcal{P}_{ph} \times \mathcal{P}_g$, let $\tilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 1, 2$, be a parametrized domain² with Lipschitz boundary $\tilde{\Gamma} = \partial\tilde{\Omega}$. We denote by $\tilde{\Gamma}_D$ and $\tilde{\Gamma}_N$ the portions of $\tilde{\Gamma}$ where Dirichlet and Neumann boundary conditions are enforced, respectively, with $\tilde{\Gamma}_D \cup \tilde{\Gamma}_N = \tilde{\Gamma}$ and $\tilde{\Gamma}_D \cap \tilde{\Gamma}_N = \emptyset$. Consider then a Hilbert space $\tilde{V} = \tilde{V}(\boldsymbol{\mu}_g) = \tilde{V}(\tilde{\Omega}(\boldsymbol{\mu}_g))$ defined over the domain $\tilde{\Omega}(\boldsymbol{\mu}_g)$, and equipped with the scalar product $(\cdot, \cdot)_{\tilde{V}}$ and the induced norm $\|\cdot\|_{\tilde{V}} = \sqrt{(\cdot, \cdot)_{\tilde{V}}}$. Furthermore, let $\tilde{V}' = \tilde{V}'(\boldsymbol{\mu}_g)$ be the dual of \tilde{V} , i.e., the space of linear and continuous functionals over \tilde{V} .

Denoting by $\tilde{G} : \tilde{V} \times \mathcal{P}_{ph} \rightarrow \tilde{V}'$ the map representing a parametrized nonlinear second-order PDE, the differential (strong) form of the problem of interest reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathcal{P}$, seek $\tilde{u}(\boldsymbol{\mu}) \in \tilde{V}(\boldsymbol{\mu}_g)$ such that

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

namely

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

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

The finite element method requires the problem (2.10) to be stated in a weak (or variational) form [40]. To this end, let us introduce the form $\tilde{g} : \tilde{V} \times \tilde{V} \times \mathcal{P} \rightarrow \mathbb{R}$, with $g(\cdot, \cdot; \boldsymbol{\mu})$ defined as:

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

The variational formulation of (2.10) then reads: given $\boldsymbol{\mu} = (\boldsymbol{\mu}_{ph}, \boldsymbol{\mu}_g) \in \mathcal{P}$, seek $\tilde{u}(\boldsymbol{\mu}) \in \tilde{V}(\boldsymbol{\mu}_g)$ such that

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

Note that the definition of \tilde{g} relies on the duality pairing $\langle \cdot, \cdot \rangle_{\tilde{V}', \tilde{V}}$ between \tilde{V} and \tilde{V}' . Hence, from the nonlinearity of \tilde{G} follows the nonlinearity of \tilde{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 3.

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 [32]. Throughout the text, we consider the following version of the parametrized Poisson equation for a state variable $\tilde{u} = \tilde{u}(\boldsymbol{\mu})$:

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

$$\tilde{u}(\boldsymbol{\mu}) = \tilde{g}(\tilde{\sigma}; \boldsymbol{\mu}_{ph}) \quad \text{on } \tilde{\Gamma}_D, \quad (2.12b)$$

$$\tilde{k}(\tilde{\sigma}, \tilde{u}(\boldsymbol{\mu}); \boldsymbol{\mu}_{ph}) \tilde{\nabla} \tilde{u}(\boldsymbol{\mu}) \cdot \tilde{\mathbf{n}} = 0 \quad \text{on } \tilde{\Gamma}_N. \quad (2.12c)$$

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

²A domain is an open and bounded set.

- $\tilde{\mathbf{x}}$ and $\tilde{\boldsymbol{\sigma}}$ denote a generic point in $\tilde{\Omega}$ and on $\tilde{\Gamma}$, respectively;
- $\tilde{\nabla}$ is the nabla operator with respect to $\tilde{\mathbf{x}}$;
- $\tilde{\mathbf{n}} = \tilde{\mathbf{n}}(\tilde{\boldsymbol{\sigma}})$ denotes the outward normal to $\tilde{\Gamma}$ in $\tilde{\boldsymbol{\sigma}}$;
- $\tilde{k} : \tilde{\Omega} \times \mathbb{R} \times \mathcal{P}_{ph} \rightarrow (0, \infty)$ is the viscosity, $\tilde{s} : \tilde{\Omega} \times \mathcal{P}_{ph} \rightarrow \mathbb{R}$ is the source term, and $\tilde{g} : \tilde{\Gamma}_D \times \mathcal{P}_{ph} \rightarrow \mathbb{R}$ encodes the Dirichlet boundary conditions; to ease the subsequent discussion, we limit the attention to homogeneous Neumann boundary constraints.

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

$$\tilde{V} = H_{\tilde{\Gamma}_D}^1(\tilde{\Omega}) = \{v \in H^1(\tilde{\Omega}) : v|_{\tilde{\Gamma}_D} = 0\},$$

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

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

where we have omitted the dependence on the space variable $\tilde{\mathbf{x}}$ for ease of notation. For the integrals in Equation (2.13) to be well-defined, we may require, for any $\boldsymbol{\mu}_g \in \mathcal{P}_g$,

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

Let then $\tilde{l} = \tilde{l}(\boldsymbol{\mu}) \in H^1(\tilde{\Omega}(\boldsymbol{\mu}_g))$ be a *lifting* function such that $\tilde{l}(\boldsymbol{\mu})|_{\tilde{\Gamma}_D} = \tilde{g}(\boldsymbol{\mu}_{ph})$, with $\tilde{g}(\boldsymbol{\mu}_{ph}) \in H^{1/2}(\tilde{\Gamma}_N)^3$. We assume that such a function can be effortlessly construct, e.g., by interpolation of the boundary condition. Hence, upon defining

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

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

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

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

Then, the weak solution of problem (2.12) is given by $\tilde{u}(\boldsymbol{\mu}) + \tilde{l}(\boldsymbol{\mu})$. 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 $v \in \tilde{V}(\boldsymbol{\mu}_g)$:

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

³We recall the definition of $H^{1/2}(\tilde{\Gamma}_N)$:

$$H^{1/2}(\tilde{\Gamma}_N) = \{v \in L^2(\tilde{\Gamma}_N) : \exists \phi \in H^1(\tilde{\Omega}) \text{ s.t. } \phi|_{\tilde{\Gamma}_N} = v\}.$$

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 $\tilde{\Omega}(\boldsymbol{\mu}_g) \subset \mathbb{R}^d$, $d = 2, 3$ [42]. Letting $\tilde{\mathbf{v}} = \tilde{\mathbf{v}}(\tilde{\mathbf{x}}; \boldsymbol{\mu})$ and $\tilde{p} = \tilde{p}(\tilde{\mathbf{x}}; \boldsymbol{\mu})$ 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:

$$\left\{ \begin{array}{ll} \tilde{\nabla} \cdot \tilde{\mathbf{v}}(\boldsymbol{\mu}) = 0 & \text{in } \tilde{\Omega}(\boldsymbol{\mu}_g), \quad (2.16a) \\ -\nu(\boldsymbol{\mu}) \tilde{\Delta} \tilde{\mathbf{v}}(\boldsymbol{\mu}) + (\tilde{\mathbf{v}}(\boldsymbol{\mu}) \cdot \tilde{\nabla}) \tilde{\mathbf{v}}(\boldsymbol{\mu}) + \frac{1}{\rho(\boldsymbol{\mu})} \tilde{\nabla} \tilde{p}(\boldsymbol{\mu}) = \mathbf{0} & \text{in } \tilde{\Omega}(\boldsymbol{\mu}_g), \quad (2.16b) \\ \tilde{\mathbf{v}}(\boldsymbol{\mu}) = \tilde{\mathbf{g}}(\boldsymbol{\mu}_{ph}) & \text{on } \tilde{\Gamma}(\boldsymbol{\mu}_g), \quad (2.16c) \\ \tilde{p}(\boldsymbol{\mu}) \tilde{\mathbf{n}} - \nu(\boldsymbol{\mu}) \tilde{\nabla} \tilde{\mathbf{v}}(\boldsymbol{\mu}) \cdot \tilde{\mathbf{n}} = \mathbf{0} & \text{on } \tilde{\Gamma}_N(\boldsymbol{\mu}_g). \quad (2.16d) \end{array} \right.$$

Here, $\tilde{\mathbf{g}}(\boldsymbol{\mu}_{ph})$ denotes the velocity field prescribed on $\tilde{\Gamma}_D$, while homogeneous Neumann conditions are applied on $\tilde{\Gamma}_N$. 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 3.18 for a practical example. For our purposes, it worths notice that the nonlinearity of problem (2.16) lies in the convective term

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

which shows up a *quadratic* nonlinearity. Conversely, both the first and third term of the momentum equation (2.16b) are linear in the solution $(\tilde{\mathbf{v}}(\boldsymbol{\mu}), \tilde{p}(\boldsymbol{\mu}))$, as Equation (2.16a), which enforces mass conservation [41].

Let us now introduce the velocity space $\tilde{X}(\boldsymbol{\mu}_g) = [H_{\tilde{\Gamma}}^1(\tilde{\Omega}(\boldsymbol{\mu}_g))]^d$ and the pressure space $\tilde{Q}(\boldsymbol{\mu}_g) = L^2(\tilde{\Omega}(\boldsymbol{\mu}_g))$ (i.e., the space of squared integrable functions defined over $\tilde{\Omega}(\boldsymbol{\mu}_g)$). A possible weak formulation for the differential problem (2.16) is given by: for a fixed $\boldsymbol{\mu} \in \mathcal{P}$, seek $(\tilde{\mathbf{v}}(\boldsymbol{\mu}), \tilde{p}(\boldsymbol{\mu})) \in \tilde{X}(\boldsymbol{\mu}_g) \times \tilde{Q}(\boldsymbol{\mu}_g)$ such that

$$\tilde{a}(\tilde{\mathbf{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \tilde{c}(\tilde{\mathbf{v}}(\boldsymbol{\mu}), \tilde{\mathbf{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \tilde{d}(\tilde{\mathbf{v}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + \tilde{b}(\tilde{p}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) = \tilde{f}_1(\boldsymbol{\chi}; \boldsymbol{\mu}), \quad (2.17a)$$

$$\tilde{b}(\xi, \tilde{\mathbf{v}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \tilde{f}_2(\xi; \boldsymbol{\mu}), \quad (2.17b)$$

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

$$\tilde{c}(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = \int_{\tilde{\Omega}(\boldsymbol{\mu}_g)} (\boldsymbol{\psi} \cdot \tilde{\nabla}) \boldsymbol{\chi} \cdot \boldsymbol{\eta} d\tilde{\Omega}(\boldsymbol{\mu}_g) \quad \forall \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in [H^1(\tilde{\Omega}(\boldsymbol{\mu}_g))]^d, \quad (2.18a)$$

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

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

$$\tilde{d}(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = \tilde{c}(\tilde{\mathbf{l}}(\boldsymbol{\mu}), \boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) + \tilde{c}(\boldsymbol{\psi}, \tilde{\mathbf{l}}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) \quad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in [H^1(\tilde{\Omega}(\boldsymbol{\mu}_g))]^d, \quad (2.18d)$$

$$\tilde{f}_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = -\tilde{a}(\tilde{\mathbf{l}}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) - \tilde{c}(\tilde{\mathbf{l}}(\boldsymbol{\mu}), \tilde{\mathbf{l}}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) \quad \forall \boldsymbol{\psi} \in \tilde{X}(\boldsymbol{\mu}_g), \quad (2.18e)$$

$$\tilde{f}_2(\xi; \boldsymbol{\mu}) = -\tilde{b}(\tilde{\mathbf{l}}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) \quad \xi \in \tilde{Q}(\boldsymbol{\mu}_g). \quad (2.18f)$$

(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 $\tilde{\mathbf{l}} = \tilde{\mathbf{l}}(\boldsymbol{\mu}) \in [H^1(\tilde{\Omega}(\boldsymbol{\mu}_g))]^d$ appearing in Equations (2.18d), (2.18e) and (2.18f) acts as a lifting function, i.e. $\tilde{\mathbf{l}}|_{\tilde{\Gamma}_D} = \tilde{\mathbf{g}}(\boldsymbol{\mu}_{ph})$, so that the weak solution for the boundary value problem (2.16) is obtained as $(\tilde{\mathbf{v}}(\boldsymbol{\mu}) + \tilde{\mathbf{l}}(\boldsymbol{\mu}), \tilde{p}(\boldsymbol{\mu}))$. In order to derive the weak formulation (2.17), we proceed as done for the Poisson equation. First multiply (2.17a) and (2.17b) by trial functions $\boldsymbol{\chi} \in \tilde{X}(\boldsymbol{\mu}_g)$ and $\xi \in \tilde{Q}(\boldsymbol{\mu}_g)$, respectively, then integrate over $\tilde{\Omega}(\boldsymbol{\mu}_g)$ and exploit integration by parts for the term

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

obtaining

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

Finally, (with a slight abuson of notation) replace $\tilde{\mathbf{v}}(\boldsymbol{\mu})$ with $\tilde{\mathbf{v}}(\boldsymbol{\mu}) + \tilde{\mathbf{l}}(\boldsymbol{\mu})$ 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 $\tilde{V}(\boldsymbol{\mu}_g) = \tilde{X}(\boldsymbol{\mu}_g) \times \tilde{Q}(\boldsymbol{\mu}_g)$ and $\tilde{\mathbf{u}}(\boldsymbol{\mu}) = (\tilde{\mathbf{v}}(\boldsymbol{\mu}), \tilde{p}(\boldsymbol{\mu})) \in \tilde{V}(\boldsymbol{\mu}_g)$, and then set

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

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

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 $\{\tilde{u}_h(\boldsymbol{\mu}^{(1)}), \dots, \tilde{u}_h(\boldsymbol{\mu}^{(N)})\}$, called *snapshots*, respectively correspondent to the parameter values $\{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$. However, recall

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

- (a) $\dim(\mathbf{u}_h(\boldsymbol{\mu}^{(j)})) = \dim(\mathbf{u}_h(\boldsymbol{\mu}^{(k)}))$, i.e. the number of degrees of freedom must be the same;
- (b) 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 $\Phi : \Omega \times \mathcal{P}_g \rightarrow \tilde{\Omega}$ such that

$$\tilde{\Omega}(\boldsymbol{\mu}_g) = \Phi(\Omega; \boldsymbol{\mu}_g). \quad (2.19)$$

As we shall prove in Sections 2.2.1 and 2.2.2, the transformation Φ 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 $G : V \times \mathcal{P} \rightarrow V'$ representing the (nonlinear) PDE over the reference domain Ω , we focus on differential problems of the form: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u(\boldsymbol{\mu}) \in V$ such that

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

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

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

where $g : V \times V \times \mathcal{P} \rightarrow \mathbb{R}$ 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 Φ , thus keeping trace of the original domain $\tilde{\Omega}(\boldsymbol{\mu}_g)$ (see Section 2.2.2). Then, the solution $\tilde{u}(\boldsymbol{\mu})$ over the original domain $\tilde{\Omega}(\boldsymbol{\mu}_g)$ can be recovered as

$$\tilde{u}(\boldsymbol{\mu}) = u(\boldsymbol{\mu}) \circ \Phi(\boldsymbol{\mu}). \quad (2.22)$$

Upon resorting to the parametrized map Φ , for any $\boldsymbol{\mu} \in \mathcal{P}$ 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 $\tilde{\Omega}_h(\boldsymbol{\mu}_g)$, given by

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

Therefore, the requirements (a) and (b) are automatically fulfilled provided that the map $\Phi(\cdot; \mu_g)$ is *conformal* for any $\mu_g \in \mathcal{P}_g$. To ensure conformality, in our numerical tests we resort to a particular choice for Φ - the boundary displacement-dependent transfinite map (BDD TM) proposed in [24] 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 $\tilde{\Omega}$ be open bounded subsets of \mathbb{R}^d , $d \geq 1$, and denote respectively by $\mathbf{x} = [x_1, \dots, x_d]^T$ and $\tilde{\mathbf{x}} = [\tilde{x}_1, \dots, \tilde{x}_d]^T$ a generic point belonging to each of them. Furthermore, assume Φ is a continuous, differentiable and invertible transformation linking Ω and $\tilde{\Omega}$,

$$\begin{aligned}\Phi : \Omega &\rightarrow \tilde{\Omega} \\ \mathbf{x} &\mapsto \tilde{\mathbf{x}} = \Phi(\mathbf{x}),\end{aligned}$$

and let Φ^{-1} be its inverse, i.e.,

$$\begin{aligned}\Phi^{-1} : \tilde{\Omega} &\rightarrow \Omega \\ \tilde{\mathbf{x}} &\mapsto \mathbf{x} = \Phi^{-1}(\tilde{\mathbf{x}}),\end{aligned}$$

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

$$\tilde{\psi}(\tilde{\mathbf{x}}) = (\psi \circ \Phi^{-1})(\tilde{\mathbf{x}}) = \psi(\Phi^{-1}(\tilde{\mathbf{x}})) = \psi(\mathbf{x}).$$

As a side remark, note that we understand the dependence of $\tilde{\Omega}$ and Φ on the parameter μ_g , since redundant for the following discussion.

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

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

Introducing the Nabla operators

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

and defining the Jacobian $\mathbb{J}_{\Phi^{-1}}(\tilde{\mathbf{x}}) : \tilde{\Omega} \rightarrow \mathbb{R}^{d \times d}$ of Φ^{-1} via

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

the compact form of (2.24) reads:

$$\tilde{\nabla} \tilde{\psi}(\tilde{\mathbf{x}}) = \mathbb{J}_{\Phi^{-1}}^T(\tilde{\mathbf{x}}) \nabla \psi(\mathbf{x}). \quad (2.25)$$

Thanks to the invertibility of Φ , the inverse function theorem [45] ensures that

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

with \mathbb{J}_{Φ} the Jacobian of Φ ,

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

Then, Equation (2.25) gets:

$$\tilde{\nabla} \tilde{\psi}(\tilde{\mathbf{x}}) = \mathbb{J}_{\Phi}^{-T}(\tilde{\mathbf{x}}) \nabla \psi(\mathbf{x}). \quad (2.26)$$

Another major ingredient we need is the change of variables formula for integrals defined over $\tilde{\Omega}$. Indeed, both formulations (2.15) and (2.17) involves integral forms. Letting $\tilde{\psi}$ be a continuous and integrable function over $\tilde{\Omega}$, we have [45]:

$$\int_{\tilde{\Omega}} \tilde{\psi}(\tilde{\mathbf{x}}) d\tilde{\Omega} = \int_{\Omega} \psi(\mathbf{x}) |\mathbb{J}_{\Phi}(\mathbf{x})| d\Omega. \quad (2.27)$$

Here, $|\mathbb{J}_{\Phi}|$ denotes the absolute value of the determinant of \mathbb{J}_{Φ} .

So far, we have considered sufficiently smooth and regular functions ψ and $\tilde{\psi}$ defined either on Ω or $\tilde{\Omega}$, respectively. However, both formulae (2.26) and (2.27) still holds when, e.g., $\psi \in H^1(\Omega)$ and $\tilde{\psi} \in H^1(\tilde{\Omega})^4$, 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 $\tilde{\psi} \in L^1(\tilde{\Omega})$. Moreover, this formula can be analogously stated also for a vectorial function $\tilde{\boldsymbol{\psi}} \in L^1(\tilde{\Omega})$ as follows:

$$\int_{\tilde{\Omega}} \tilde{\boldsymbol{\psi}} d\tilde{\Omega} = \int_{\Omega} \boldsymbol{\psi} |\mathbb{J}_{\Phi}| d\Omega. \quad (2.28)$$

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):

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

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

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

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

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

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 $V = H_{\Gamma_D}^1(\Omega)$ 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 $\boldsymbol{\mu} \in \mathcal{P}$, seek $u(\boldsymbol{\mu}) \in V$ such that

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

with

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

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

for any $w, v \in V$ and $\boldsymbol{\mu} \in \mathcal{P}$. 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 $k(\mathbf{x}, \cdot; \boldsymbol{\mu}) = \tilde{k}(\Phi(\mathbf{x}; \boldsymbol{\mu}), \cdot; \boldsymbol{\mu})$ is the diffusion coefficient, $s(\mathbf{x}; \boldsymbol{\mu}) = \tilde{s}(\Phi(\mathbf{x}; \boldsymbol{\mu}); \boldsymbol{\mu})$ is the source term, and $g(\mathbf{x}; \boldsymbol{\mu}) = \tilde{g}(\Phi(\mathbf{x}; \boldsymbol{\mu}); \boldsymbol{\mu})$ represents the state field prescribed on Γ_D . Then, in (2.34) we resort to a lifting function $l(\boldsymbol{\mu}) \in H^1(\Omega)$ with $l(\boldsymbol{\mu})|_{\Gamma_D} = g(\boldsymbol{\mu})$ such that the weak solution to the problem (2.12) re-stated over Ω is obtained as $u(\boldsymbol{\mu}) + l(\boldsymbol{\mu})$.

Similarly, the variational formulation over Ω of the boundary value problem (2.16) for the stationary incompressible Navier-Stokes equations follows from (2.17), (2.36), (2.30), (2.31) and (2.32). Consider the velocity space $X = [H_{\Gamma_D}^1(\Omega)]^d$ and the pressure space $Q = L^2(\Omega)$, and set $V = X \times Q$. Then, given $\boldsymbol{\mu} \in \mathcal{P}$, the problem consists in finding $u(\boldsymbol{\mu}) = (\mathbf{v}(\boldsymbol{\mu}), p(\boldsymbol{\mu})) \in V$ so that

$$a(\mathbf{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\mathbf{v}(\boldsymbol{\mu}), \mathbf{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + d(\mathbf{v}(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}) + b(p(\boldsymbol{\mu}), \nabla \cdot \boldsymbol{\chi}; \boldsymbol{\mu}) = f_1(\boldsymbol{\chi}; \boldsymbol{\mu}), \quad (2.35a)$$

$$b(\nabla \cdot \mathbf{v}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) = f_2(\xi; \boldsymbol{\mu}), \quad (2.35b)$$

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

$$c(\boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta}; \boldsymbol{\mu}) = \int_{\Omega} (\boldsymbol{\psi} \cdot \mathbb{J}_{\Phi}^{-T}(\boldsymbol{\mu}) \nabla) \boldsymbol{\chi} \cdot \boldsymbol{\eta} |\mathbb{J}_{\Phi}(\boldsymbol{\mu})| d\Omega \quad \forall \boldsymbol{\psi}, \boldsymbol{\chi}, \boldsymbol{\eta} \in [H^1(\Omega)]^d, \quad (2.36a)$$

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

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

$$d(\boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) = c(l(\boldsymbol{\mu}), \boldsymbol{\psi}, \boldsymbol{\chi}; \boldsymbol{\mu}) + c(\boldsymbol{\psi}, l(\boldsymbol{\mu}), \boldsymbol{\chi}; \boldsymbol{\mu}); \quad \forall \boldsymbol{\psi}, \boldsymbol{\chi} \in X, \quad (2.36d)$$

$$f_1(\boldsymbol{\psi}; \boldsymbol{\mu}) = -a(\mathbf{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) - c(\mathbf{l}(\boldsymbol{\mu}), \mathbf{l}(\boldsymbol{\mu}), \boldsymbol{\psi}; \boldsymbol{\mu}) \quad \forall \boldsymbol{\psi} \in X, \quad (2.36e)$$

$$f_2(\xi; \boldsymbol{\mu}) = -b(\mathbf{l}(\boldsymbol{\mu}), \xi; \boldsymbol{\mu}) \quad \forall \xi \in Q. \quad (2.36f)$$

In the above equations, $\mathbf{l}(\boldsymbol{\mu}) \in [H^1(\Omega)]^d$ denotes the lifting vector field, with $\mathbf{l}(\boldsymbol{\mu})|_{\Gamma_D} = \mathbf{g}(\boldsymbol{\mu})$, $\mathbf{g}(\mathbf{x}; \boldsymbol{\mu}) = \tilde{\mathbf{g}}(\Phi(\mathbf{x}; \boldsymbol{\mu}); \boldsymbol{\mu})$ being the velocity field prescribed on Γ_D . Then, the weak solution to (2.16) defined over the fixed domain Ω is given by $(\mathbf{v}(\boldsymbol{\mu}) + \mathbf{l}(\boldsymbol{\mu}), p(\boldsymbol{\mu}))$.

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 displacement-dependent transfinite map (BDD TM) to construct the bijection $\Phi(\boldsymbol{\mu}_g)$ from the reference domain Ω to the class of parametrized domains $\{\tilde{\Omega}(\boldsymbol{\mu}_g), \boldsymbol{\mu}_g \in \mathcal{P}_g\}$. BDD TM has been proposed in [24] 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 $\tilde{\Omega}(\boldsymbol{\mu}_g)$ be *curved polygonal* two-dimensional domains with the same number n of edges. Denote then by Γ_i , $i = 1, \dots, n$, the i -th edge of Ω , and by $\tilde{\Gamma}_i(\boldsymbol{\mu}_g)$ the correspondent edge of $\tilde{\Omega}(\boldsymbol{\mu}_g)$. For both the reference and the deformed domain, suppose the edges are ordered clockwise and assume they admit parametrizations of the form:

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

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

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* $\mathbf{d}_i : [0, 1] \times \mathcal{P}_g \rightarrow \mathbb{R}^2$ defined as:

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

For each point on Γ_i , this function gives us the relative displacement between the new and the old position of the boundary [24]. Then, the displacement of any internal point of Ω is sought as a linear combination of the boundary displacement functions $\{\mathbf{d}_i\}_{i=1}^n$. For this purpose, each edge Γ_i is associated with two scalar-valued functions - a *weight function* ξ_i and a *projection function* π_i . As we shall see 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 , $i = 1, \dots, n$, the weight function ξ_i acts as multiplier

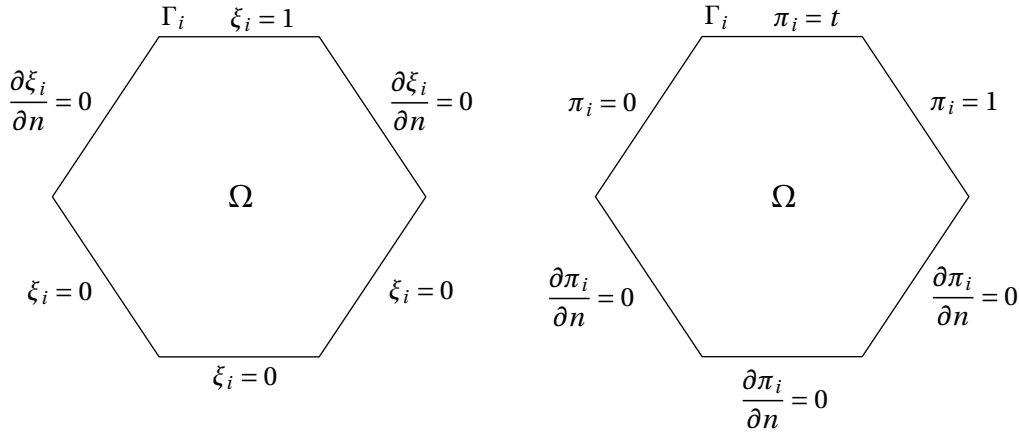


Figure 2.1. Representation of the boundary conditions for the Laplace problems (2.38) (left) and (2.39) (right) stated on a hexagonal reference domain Ω .

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

$$\left\{ \begin{array}{ll} \Delta \xi_i = 0 & \text{in } \Omega, \\ \xi_i = 1 & \text{on } \Gamma_i, \\ \frac{\partial \xi_i}{\partial n} = 0 & \text{on } \Gamma_j, j = i-1, i+1, \\ \xi_i = 0 & \text{on } \Gamma_j, j \neq i-1, i, i+1. \end{array} \right. \quad (2.38)$$

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:

$$\left\{ \begin{array}{ll} \Delta \pi_i = 0 & \text{in } \Omega, \\ \pi_i = t & \text{on } \Gamma_i, \\ \pi_i = 0 & \text{on } \Gamma_{i-1}, \\ \pi_i = 1 & \text{on } \Gamma_{i+1}, \\ \frac{\partial \pi_i}{\partial n} = 0 & \text{on } \Gamma_j, j \neq i-1, i, i+1. \end{array} \right. \quad (2.39)$$

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 $\mathbf{x} = (x, y)$, we have:

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

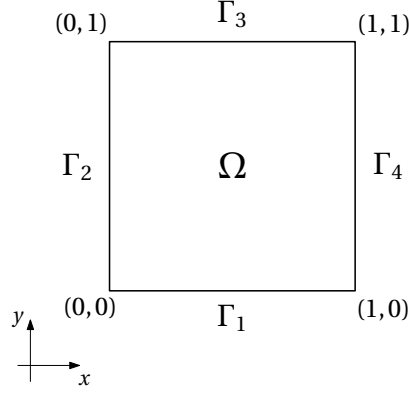


Figure 2.2. Clockwise enumeration for the edges of the reference squared domain Ω used in the numerical tests. The coordinates of the vertices are reported too.

Finally, given any parameter vector $\boldsymbol{\mu} \in \mathcal{P}_g$, the boundary displacement-dependent transfinite map $\Phi : \Omega \times \mathcal{P}_g \rightarrow \tilde{\Omega}$ is constructed as [24]:

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

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) [45]. In our case, plugging (2.40) into (2.41) yields:

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

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 [24]. Moreover, in the numerical simulations we performed the map (2.42) yielded regular transformed grids $\tilde{\Omega}_h(\boldsymbol{\mu}_g)$ on $\tilde{\Omega}(\boldsymbol{\mu}_g)$. In particular, for any tested value of $\boldsymbol{\mu}_g \in \mathcal{P}$, the mesh $\tilde{\Omega}_h(\boldsymbol{\mu}_g)$ 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 $\Phi(\boldsymbol{\mu}_g)$. 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 $\boldsymbol{\mu} \in \mathcal{P}$, the viscosity $k(\cdot, \cdot; \boldsymbol{\mu}) : \Omega \times \mathbb{R} \rightarrow [0, \infty)$ is twice continuously differentiable with respect to its second argument, and, for any compact set $\Omega_c \subset \Omega$ and any bounded interval I :

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

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., [6].

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 [41]. Whereas, uniqueness is guaranteed upon a *small data* hypothesis on the boundary conditions and a possible forcing term [11].

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., [40].

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 $\boldsymbol{\mu} \in \mathcal{P}$, find $u_h(\boldsymbol{\mu}) \in V_h$ such that

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

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 $G : V \times \mathcal{P} \rightarrow V'$ representing our (nonlinear) differential operator, and denote by $D_u G(z, \boldsymbol{\mu}) : V \rightarrow V'$ its partial Frechét derivative at $(z, \boldsymbol{\mu}) \in V \times \mathcal{P}$, i.e., a linear bounded operator such that [45]

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

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

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

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

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

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

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

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.,

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

Hence, denoting by $\mathbf{u}_h(\boldsymbol{\mu}) \in \mathbb{R}^M$ the vector collecting the *degrees of freedom* $\{u_h^{(j)}(\boldsymbol{\mu})\}_{j=1}^M$ and exploiting the linearity of $g(\cdot, \cdot; \boldsymbol{\mu})$ in the second argument, the problem (2.43) is equivalent to: given $\boldsymbol{\mu} \in \mathcal{P}$, find $\mathbf{u}_h \in \mathbb{R}^M$ such that

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

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

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

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

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

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

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

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

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

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

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

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 [41]. Therefore, an approximation to the unknown field

$$u = (u_1, \dots, u_S) \in V = V_1 \times \dots \times V_S \subset [H^1(\Omega)]^S \quad (2.53)$$

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

$$V_h = (V_1 \cap X_h^{r_1}) \times \dots \times (V_S \cap X_h^{r_S}). \quad (2.54)$$

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 $\{\mathbf{N}_i\}_{i=1}^M$, called *nodes*, which usually form a superset of the *vertices* of Ω_h . Upon requiring that

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

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

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

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).

```

1: function  $\mathbf{u}_h = \text{NEWTON}(\boldsymbol{\mu}, \Omega_h, \mathbf{u}_h^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta, K_{max})$ 
2:   build the transformation map  $\Phi(\mathbf{x}; \boldsymbol{\mu})$  via Equation (2.41)
3:    $k = 0$ 
4:   do
5:     evaluate the residual vector  $\mathbf{G}_h(\mathbf{u}_h^k; \boldsymbol{\mu})$ 
6:     form the Jacobian  $\mathbb{J}_h(\mathbf{u}_h^k; \boldsymbol{\mu})$ 
7:     solve  $\mathbb{J}_h(\mathbf{u}_h^k; \boldsymbol{\mu}) \delta \mathbf{u}_h^k = -\mathbf{G}_h(\mathbf{u}_h^k; \boldsymbol{\mu})$ 
8:     set  $\mathbf{u}_h^{k+1} = \mathbf{u}_h^k + \delta \mathbf{u}_h^k$ 
9:      $k \leftarrow k + 1$ 
10:  while  $k < K_{max}$  and  $\|\mathbf{G}_h(\mathbf{u}_h^k)\| > \delta$ 
11:   $\mathbf{u}_h = \mathbf{u}_h^k$ 
12: 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

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

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 argument. Then, introducing the discrete lifting function $l_h(\mathbf{x}; \boldsymbol{\mu}) \in X_h^1$, obtained via, e.g., interpolation of the boundary conditions over the linear finite element space X_h^1 , for any $\boldsymbol{\mu} \in \mathcal{P}$ the residual vector $\mathbf{G}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu})$ reads

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

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

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

for any $i, j = 1, \dots, M$. 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 general nonlinearity of the diffusion coefficient k and the transformation map Φ . 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 stability condition (2.45) [42], which for the particular case of the Navier-Stokes equations can be expressed as

$$\beta(\boldsymbol{\mu}) = \inf_{q_h \in Q_h} \sup_{w_h \in X_h} \frac{b_h(q_h, w_h; \boldsymbol{\mu})}{\|w_h\|_X \|v_h\|_Q} \geq \beta_0 \quad \forall \boldsymbol{\mu} \in \mathcal{P}, \quad (2.55)$$

where $b_h(\cdot, \cdot; \boldsymbol{\mu})$ represents a suitable discretization of the bilinear form $b(\cdot, \cdot; \boldsymbol{\mu})$ (see Equation (2.36c)). 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 than that of the pressure field [40]. 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 [38]. Therefore, according to the notation introduced in (2.53) and (2.54), we have

$$u = (v_x, v_y, p) \quad \text{and} \quad V_h = \left(X_h^2 \cap H_{\Gamma_D}^1(\Omega) \right) \times \left(X_h^2 \cap H_{\Gamma_D}^1(\Omega) \right) \times \left(X_h^1 \cap L^2(\Omega) \right). \quad (2.56)$$

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

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

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

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

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

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

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 $j = 1, \dots, M_n$,

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

so that

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

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):

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

with

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

where $a_K(\cdot, \cdot; \boldsymbol{\mu})$, $d_K(\cdot, \cdot; \boldsymbol{\mu})$, $c_K(\cdot, \cdot, \cdot; \boldsymbol{\mu})$, $b_K(\cdot, \cdot; \boldsymbol{\mu})$, $f_{1K}(\cdot; \boldsymbol{\mu})$ and $f_{2K}(\cdot; \boldsymbol{\mu})$ represent the restrictions of $a(\cdot, \cdot; \boldsymbol{\mu})$, $d(\cdot, \cdot; \boldsymbol{\mu})$, $c(\cdot, \cdot, \cdot; \boldsymbol{\mu})$, $b(\cdot, \cdot; \boldsymbol{\mu})$, $f_1(\cdot; \boldsymbol{\mu})$ and $f_2(\cdot; \boldsymbol{\mu})$, respectively, over the element $K \in \Omega_h$. Moreover, we recall that the definitions of the bilinear form $d(\cdot, \cdot; \boldsymbol{\mu})$ and the linear forms $f_1(\cdot; \boldsymbol{\mu})$ and $f_2(\cdot; \boldsymbol{\mu})$ rely on the introduction of a lifting vector $\mathbf{l} = \mathbf{l}(\boldsymbol{\mu}) \in [H^1(\Omega)]^2$, which has then to be interpolated over the finite element space $[X_h^2]^2$, yielding the discrete lifting vector $\mathbf{l}_h = \mathbf{l}_h \in [X_h^2]^2$.

Upon introducing the matrices $\mathbb{A}(\boldsymbol{\mu}) \in \mathbb{R}^{2M_n \times 2M_n}$, $\mathbb{C}(\mathbf{v}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) \in \mathbb{R}^{2M_n \times 2M_n}$, $i = 1, \dots, 2M_n$, and $\mathbb{B}(\boldsymbol{\mu}) \in \mathbb{R}^{M_v \times 2M_n}$, and the vectors $\mathbf{f}_1(\boldsymbol{\mu}) \in \mathbb{R}^{2M_n}$ and $\mathbf{f}_2(\boldsymbol{\mu}) \in \mathbb{R}^{M_v}$, defined as

$$(\mathbb{A}(\boldsymbol{\mu}))_{i,j} = a_h(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i; \boldsymbol{\mu}) + d_h(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i; \boldsymbol{\mu}) \quad i, j = 1, \dots, 2M_n, \quad (2.57a)$$

$$(\mathbb{C}(\mathbf{v}_h(\boldsymbol{\mu}); \boldsymbol{\mu}))_{i,j} = \sum_{k=1}^{2M_n} v_h^{(k)} c_h(\boldsymbol{\phi}_j, \boldsymbol{\phi}_k, \boldsymbol{\phi}_i; \boldsymbol{\mu}) \quad i, j = 1, \dots, 2M_n, \quad (2.57b)$$

$$(\mathbb{B}(\boldsymbol{\mu}))_{q,j} = b_h(\boldsymbol{\phi}_q^p, \boldsymbol{\phi}_j; \boldsymbol{\mu}) \quad q = 1, \dots, M_v, j = 1, \dots, 2M_n, \quad (2.57c)$$

$$(\mathbf{f}_1(\boldsymbol{\mu}))_i = f_{1h}(\boldsymbol{\phi}_i; \boldsymbol{\mu}) \quad i = 1, \dots, 2M_n, \quad (2.57d)$$

$$(\mathbf{f}_2(\boldsymbol{\mu}))_q = f_{2h}(\boldsymbol{\phi}_q^p; \boldsymbol{\mu}) \quad q = 1, \dots, M_v, \quad (2.57e)$$

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

$$\begin{bmatrix} \mathbb{A}(\boldsymbol{\mu}) + \mathbb{C}(\mathbf{v}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) & \mathbb{B}(\boldsymbol{\mu})^T \\ \mathbb{B}(\boldsymbol{\mu}) & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_h(\boldsymbol{\mu}) \\ \mathbf{p}_h(\boldsymbol{\mu}) \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1(\boldsymbol{\mu}) \\ \mathbf{f}_2(\boldsymbol{\mu}) \end{bmatrix}. \quad (2.58)$$

Hence, the Jacobian is given by:

$$\mathbb{J}_h(\mathbf{u}_h(\boldsymbol{\mu}); \boldsymbol{\mu}) = \begin{bmatrix} \mathbb{A}(\boldsymbol{\mu}) + (\bar{\mathbb{C}} + \bar{\mathbb{C}}^T) \mathbf{v}_h(\boldsymbol{\mu}) & \mathbb{B}^T(\boldsymbol{\mu}) \\ \mathbb{B}(\boldsymbol{\mu}) & 0 \end{bmatrix}, \quad (2.59)$$

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

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

2.5 POD-Galerkin 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 $\boldsymbol{\mu} \in \mathcal{P}$ [10]. 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 [21, 41].

As widely anticipated, in this work we focus on reduced basis (RB) methods for the approximation of the variational problem (2.21). To this end, let us recall the definition of the solution manifold \mathcal{M} ,

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

and its discrete counterpart \mathcal{M}_h ,

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

For any $\boldsymbol{\mu} \in \mathcal{P}$, 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 $u(\boldsymbol{\mu})$ (either refining the computational mesh or increasing the order of the FE space), so that \mathcal{M}_h provides a good approximation of \mathcal{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 $\{\psi_1, \dots, \psi_L\} \subset V_h$, called *reduced basis functions*, built from a collection of high-fidelity solutions $\{u_h(\boldsymbol{\mu}^{(1)}), \dots, u_h(\boldsymbol{\mu}^{(N)})\} \subset \mathcal{M}_h$, called *snapshots*, where the discrete and finite set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathcal{P}$ may consist of either a uniform lattice or randomly generated points over the parameter domain \mathcal{P} [21]. 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., [7]). Here, we pursue the first approach, employing the well-known Proper Orthogonal Decomposition (POD) method [47], detailed in the following subsection.

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

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

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

$$u_L(\mathbf{x}; \boldsymbol{\mu}) = \sum_{l=1}^L u_{\text{rb}}^{(l)}(\boldsymbol{\mu}) \psi_l(\mathbf{x}) \in V_{\text{rb}}, \quad (2.60)$$

with $\mathbf{u}_{\text{rb}}(\boldsymbol{\mu}) = [u_{\text{rb}}^{(1)}(\boldsymbol{\mu}), \dots, u_{\text{rb}}^{(L)}(\boldsymbol{\mu})]^T \in \mathbb{R}^L$ 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 rationale 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_{rb} features a dimensionality which is significantly smaller than the dimension of the original finite element space V_h ; in other terms, $L \ll M$ must hold. However, this tacitly assumes that the solution manifold $\mathcal{M} \subset V$ (or, equivalently, $\mathcal{M}_h \subset V_h$, for what previously said) is actually of low-dimension, and can then be accurately approximated by a subspace of reduced dimension L [21]. To further investigate this necessary hypothesis, it is worth introducing the notion of Kolmogorov L -width. Here, we report the definition provided in [29].

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

$$E(A; X_L) = \sup_{x \in A} \inf_{y \in X_L} \|x - y\|_X. \quad (2.61)$$

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

$$\begin{aligned} d_L(A, X) &= \inf \{E(A; X_L) : X_L \text{ is an } L\text{-dimensional subspace of } X\} \\ &= \inf_{X_L} \sup_{x \in A} \inf_{y \in X_L} \|x - y\|_X. \end{aligned} \quad (2.62)$$

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 [29]. 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 $A = \mathcal{M}$, 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 [4].

Nevertheless, we still have to ensure that the RB space V_{rb} , i.e., the chosen X_L in (2.62), actually attains the infimum $d_L(\mathcal{M}, V)$, 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 $u_{\text{rb}}(\boldsymbol{\mu})$:

$$\|u(\boldsymbol{\mu}) - u_{\text{rb}}(\boldsymbol{\mu})\|_V \leq \|u(\boldsymbol{\mu}) - u_h(\boldsymbol{\mu})\|_V + \|u_h(\boldsymbol{\mu}) - u_{\text{rb}}(\boldsymbol{\mu})\|_V \quad \forall \boldsymbol{\mu} \in \mathcal{P}.$$

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 $\|u_h(\boldsymbol{\mu}) - u_{\text{rb}}(\boldsymbol{\mu})\|_V$, i.e., the error between the truth and the reduced solution. In this respect, the last decade has witnessed the development of different *a priori* and *a posteriori* estimates for such an error (see, e.g., [4, 21, 29]), 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., [10, 41]), 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 $\|u_h(\boldsymbol{\mu}) - u_{\text{rb}}(\boldsymbol{\mu})\|_V$ on a finite and discrete *test* dataset $\Xi_{te} \in \mathcal{P}$. 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 $u_{\text{rb}}(\boldsymbol{\mu})$, whose general form is given in (2.60), we proceed to project the variational problem (2.21) onto the RB space V_{rb} pursuing a standard Galerkin approach, leading to the following *reduced basis problem*: given $\boldsymbol{\mu} \in \mathcal{P}$, find $u_{\text{rb}}(\boldsymbol{\mu}) \in V_{\text{rb}}$ so that

$$g(u_{\text{rb}}(\boldsymbol{\mu}), v_{\text{rb}}; \boldsymbol{\mu}) = 0 \quad \forall v_{\text{rb}} \in V_{\text{rb}}. \quad (2.63)$$

Then, the Newton's method applied to (2.63) entails, at each iteration $k \geq 0$, the resolution of the following linearized problem: given $\boldsymbol{\mu} \in \mathcal{P}$, seek $\delta u_{\text{rb}}^k(\boldsymbol{\mu})$ such that

$$dg[u_{\text{rb}}^k(\boldsymbol{\mu})](\delta u_{\text{rb}}^k(\boldsymbol{\mu}), v_{\text{rb}}; \boldsymbol{\mu}) = -g(u_{\text{rb}}^k(\boldsymbol{\mu}), v_{\text{rb}}; \boldsymbol{\mu}) \quad \forall v_{\text{rb}} \in V_{\text{rb}}, \quad (2.64)$$

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

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 $\mathbb{V} = [\boldsymbol{\psi}_1, \dots, \boldsymbol{\psi}_L] \in \mathbb{R}^{M \times L}$. For any $v_{\text{rb}} \in V_{\text{rb}}$, the matrix \mathbb{V} encodes the change of variables from the RB basis to the standard FE basis, i.e.,

$$\mathbf{v}_L = \mathbb{V} \mathbf{v}_{\text{rb}}. \quad (2.65)$$

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

- $\mathbf{v}_{\text{rb}} \in \mathbb{R}^L$, collecting the coefficients for the expansion of v_{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_{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_{rb} .

Upon choosing $v_{\text{rb}} = \psi_l$, $l = 1, \dots, L$, in the RB problem (2.63), for any $\boldsymbol{\mu} \in \mathcal{P}$ we get the following set of equations:

$$g(u_{\text{rb}}(\boldsymbol{\mu}), \psi_l; \boldsymbol{\mu}) = 0 \quad 1 \leq l \leq L. \quad (2.66)$$

Plugging into (2.66) the expansion of ψ_l , $l = 1, \dots, L$, in terms of the canonical FE basis $\{\phi_m\}_{m=1}^M$, i.e.,

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

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

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

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 (2.65). Then, the algebraic formulation of the reduced basis problem (2.63) can be written in compact form as:

$$\mathbf{G}_{\text{rb}}(\mathbf{u}_{\text{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbb{V} \mathbf{u}_{\text{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) = \mathbf{0} \in \mathbb{R}^L. \quad (2.67)$$

Hence, the *reduced nonlinear system* (2.67) imposes the orthogonality (in the Euclidean scalar product) of the residual vector $\mathbf{G}_h(\cdot; \boldsymbol{\mu})$, evaluated in $\mathbb{V} \mathbf{u}_{\text{rb}}(\boldsymbol{\mu})$, 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 $\mathbb{J}_{\text{rb}}(\cdot; \boldsymbol{\mu})$ of $\mathbf{G}_{\text{rb}}(\cdot; \boldsymbol{\mu})$ is given by

$$\mathbb{J}_{\text{rb}}(\mathbf{w}; \boldsymbol{\mu}) = \mathbb{V}^T \mathbb{J}_h(\mathbb{V} \mathbf{w}; \boldsymbol{\mu}) \mathbb{V} \in \mathbb{R}^{L \times L} \quad \forall \mathbf{w} \in \mathbb{R}^L, \forall \boldsymbol{\mu} \in \mathcal{P}, \quad (2.68)$$

so that, starting from an initial guess $\mathbf{u}_{\text{rb}}^0 \in \mathbb{R}^L$, each iteration k , $k \geq 0$, of the Newton's method applied to the reduced nonlinear system (2.67) entails the resolution of the linear system

$$\mathbb{J}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k(\boldsymbol{\mu}); \boldsymbol{\mu}) \delta \mathbf{u}_{\text{rb}}^k(\boldsymbol{\mu}) = -\mathbf{G}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k(\boldsymbol{\mu}); \boldsymbol{\mu}), \quad (2.69)$$

with $\mathbf{u}_{\text{rb}}^{k+1}(\boldsymbol{\mu}) = \mathbf{u}_{\text{rb}}^k(\boldsymbol{\mu}) + \delta \mathbf{u}_{\text{rb}}^k(\boldsymbol{\mu})$. 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_{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 [41]. 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 (2.67).

```

1: function  $\mathbf{u}_{\text{rb}} = \text{NEWTONRB}(\boldsymbol{\mu}, \Omega_h, \mathbb{V}, \mathbf{u}_{\text{rb}}^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta, K_{\max})$ 
2:   build the transformation map  $\Phi(\mathbf{x}; \boldsymbol{\mu})$  via Equation (2.41)
3:    $k = 0$ 
4:   do
5:     set  $\mathbf{u}_L^k = \mathbb{V} \mathbf{u}_{\text{rb}}^k$ 
6:     evaluate the reduced residual vector  $\mathbf{G}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k; \boldsymbol{\mu}) = \mathbb{V}^T \mathbf{G}_h(\mathbf{u}_L^k; \boldsymbol{\mu})$ 
7:     form the Jacobian  $\mathbb{J}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k; \boldsymbol{\mu}) = \mathbb{V}^T \mathbb{J}_h(\mathbf{u}_L^k; \boldsymbol{\mu}) \mathbb{V}$ 
8:     solve  $\mathbb{J}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k; \boldsymbol{\mu}) \delta \mathbf{u}_{\text{rb}}^k = -\mathbf{G}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k; \boldsymbol{\mu})$ 
9:     set  $\mathbf{u}_{\text{rb}}^{k+1} = \mathbf{u}_{\text{rb}}^k + \delta \mathbf{u}_{\text{rb}}^k$ 
10:     $k \leftarrow k + 1$ 
11:  while  $k < K_{\max}$  and  $\|\mathbf{G}_{\text{rb}}(\mathbf{u}_{\text{rb}}^k)\| > \delta$ 
12:   $\mathbf{u}_{\text{rb}} = \mathbf{u}_{\text{rb}}^k$ 
13: end function

```

the orthonormal basis to retain only a few POD modes, then projecting the system onto the truncated basis [47]. 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 [19].

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 $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathcal{P}$, and let \mathcal{M}_{Ξ_N} be the subspace spanned by the snapshots, i.e.,

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

Clearly, $\mathcal{M}_{\Xi_N} \subset \mathcal{M}_h$ and we can assume that \mathcal{M}_{Ξ_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}_{Ξ_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

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

constitutes a low-rank approximation of \mathcal{M}_{Ξ_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.,

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

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

$$\mathbb{S} = \mathbb{W} \begin{bmatrix} \mathbb{D} & 0 \\ 0 & 0 \end{bmatrix} \mathbb{Z}^T = \mathbb{W} \Sigma \mathbb{Z}^T, \quad (2.70)$$

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

$$\mathbb{S}\mathbb{S}^T\mathbf{w}_m = \begin{cases} \sigma_m^2 \mathbf{w}_m & \text{for } 1 \leq m \leq R, \\ \mathbf{0} & \text{for } R+1 \leq m \leq M, \end{cases} \quad (2.71)$$

$$\mathbb{S}^T\mathbb{S}\mathbf{z}_n = \begin{cases} \sigma_n^2 \mathbf{z}_n & \text{for } 1 \leq n \leq R, \\ \mathbf{0} & \text{for } R+1 \leq n \leq N, \end{cases} \quad (2.72)$$

$$\mathbb{S}\mathbf{z}_i = \sigma_i \mathbf{w}_i \quad \text{for } 1 \leq i \leq R, \quad (2.73)$$

$$\mathbb{S}^T\mathbf{w}_i = \sigma_i \mathbf{z}_i \quad \text{for } 1 \leq i \leq R. \quad (2.74)$$

In particular, Equations (2.71) and (2.72) 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 (2.70), the SVD of \mathbb{S} can be cast in the compact form:

$$\mathbb{S} = \mathbb{W}^R \mathbb{D} (\mathbb{Z}^R)^T, \quad (2.75)$$

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 \mathbb{W} and \mathbb{Z} , respectively, i.e.,

$$\mathbb{W}^R = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_R] \quad \text{and} \quad \mathbb{Z}^R = [\mathbf{z}_1 \mid \dots \mid \mathbf{z}_R]. \quad (2.76)$$

Letting $\mathbb{B}^R = \mathbb{D} (\mathbb{Z}^R)^T \in \mathbb{R}^{R \times N}$ and exploiting the orthonormality of the columns of \mathbb{W}^R , the generic column $\mathbf{s}_n = \mathbf{u}_h(\boldsymbol{\mu}^{(n)})$ of \mathbb{S} , $n = 1, \dots, N$, can be expressed as [47]:

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

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 $\text{Col}(\mathbb{S})$ of \mathbb{S} . Moreover, from the above calculations follow that $\mathbb{B}_{r,n}^R = (\mathbf{s}_n, \mathbf{w}_r)_{\mathbb{R}^M}$ 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 $\{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_L\}$, with $L < R$. It is an easy matter to show that for each \mathbf{s}_n , $n = 1, \dots, N$, the element of $\text{span}\{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_L\}$ closest to \mathbf{s}_n in the Euclidean norm $\|\cdot\|_{\mathbb{R}^M}$ is given by

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

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

$$\varepsilon(\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_L) = \sum_{n=1}^N \left\| \mathbf{s}_n - \sum_{l=1}^L (\mathbf{s}_n, \tilde{\mathbf{w}}_l)_{\mathbb{R}^M} \tilde{\mathbf{w}}_l \right\|_{\mathbb{R}^M}^2. \quad (2.77)$$

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

Theorem 2.5.1 (Schmidt-Eckart-Young). *Consider a rectangular matrix $\mathbb{S} = [\mathbf{s}_1 \mid \dots \mid \mathbf{s}_N] \in \mathbb{R}^{M \times N}$ 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 $\mathbb{W} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_M] \in \mathbb{R}^{M \times M}$ and $\mathbb{Z} = [\mathbf{z}_1 \mid \dots \mid \mathbf{z}_N] \in \mathbb{R}^{N \times N}$ orthogonal matrices, and $\Sigma \in \mathbb{R}^{M \times N}$ defined as in (2.70). Further, let $\mathbb{S} = \mathbb{W}^R \mathbb{D} (\mathbb{Z}^R)^T = \mathbb{W}^R \mathbb{B}^R$ be the compact form of the SVD of \mathbb{S} , with $\mathbb{W}^R \in \mathbb{R}^{M \times R}$ and $\mathbb{Z}^R \in \mathbb{R}^{N \times R}$ defined as in (2.76), $\mathbb{D} = \text{diag}(\sigma_1, \dots, \sigma_R) \in \mathbb{R}^{R \times R}$, and $\mathbb{B}^R = \mathbb{D} (\mathbb{Z}^R)^T \in \mathbb{R}^{R \times N}$. Suppose that $\widehat{\mathbb{W}}^R \in \mathbb{R}^{M \times R}$ denotes a matrix with pairwise orthonormal 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

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

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

$$\|\mathbb{S} - \mathbb{W}^L \mathbb{B}^L\|_F \leq \|\mathbb{S} - \widehat{\mathbb{W}}^L \mathbb{C}^L\|_F. \quad (2.78)$$

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

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

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}) [47].

Regarding (2.78), let us note that

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

Then, according to (2.77),

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

and, analogously,

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

Hence, the optimality condition (2.78) 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., [47])

$$\varepsilon(\mathbf{w}_1, \dots, \mathbf{w}_L) = \sum_{j=L+1}^R \sigma_j^2, \quad (2.79)$$

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

$$\mathbb{V} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_L]^5. \quad (2.80)$$

Hence, in the reduced basis problem (2.63) the test and trial functions are picked from the subspace $V_{\text{rb}} = V_{\text{POD}}$ of V_h , spanned by the FE functions $\{\psi_l\}_{l=1}^L$, given by

$$\psi_l(\mathbf{x}) = \sum_{m=1}^M \psi_l^{(m)} \phi_m(\mathbf{x}) = \sum_{m=1}^M (\mathbf{w}_l)_m \phi_m(\mathbf{x}) \quad \text{for } 1 \leq l \leq L. \quad (2.81)$$

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 $(\cdot, \cdot)_h$:

$$(\chi_h, \xi_h)_h = \sum_{i=1}^M \chi_h(\mathbf{N}_i) \xi_h(\mathbf{N}_i) = \sum_{i=1}^M \chi_h^{(i)} \xi_h^{(i)} = (\boldsymbol{\chi}_h, \boldsymbol{\xi}_h)_{\mathbb{R}^M}. \quad (2.82)$$

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

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

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

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

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

then by (2.73) we have

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

Let us conclude by further discussing the estimate (2.79) 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 [21]. 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 [5]. In particular, the minimum number of basis functions ensuring a POD error smaller

⁵Please observe that generally we do not decorate \mathbb{V} with any subscript and superscript reporting the dimension of the reduced basis it represents, unless not clear from the context.

than a desired tolerance δ corresponds with the smallest integer L satisfying the following inequality:

$$\frac{\sum_{l=1}^L \sigma_l^2}{\sum_{l=1}^R \sigma_l^2} > 1 - \delta. \quad (2.83)$$

In other terms, the energy retained by the first L POD modes must be greater than the fraction $1 - \delta$ of the total energy of the snapshots [41]. This criterium, also known as *relative information content*, has been embodied into Algorithm 2.3 summarizing the POD method. Moreover, it also implies that

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

However, (2.79), then (2.83), 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 \mathcal{M}_{Ξ_N} ⁶, we do not have any guarantee for all the other elements in the discrete solution manifold \mathcal{M}_h . Therefore, one may need a large number of snapshots, so ensuring that \mathcal{M}_{Ξ_N} provides a good approximation of \mathcal{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 (2.79) on a discrete and finite parameter test set $\Xi_{te} \subset \mathcal{P}$, with $\Xi_{te} \cap \Xi_N = \emptyset$.

Algorithm 2.3 The POD algorithm.

```

1: function  $\mathbb{V} = \text{POD}(\mathbb{S}, \delta)$ 
2:   if  $M \leq N$  then
3:      $\mathbb{M} = \mathbb{S}^T \mathbb{S}$ 
4:     for  $i = 1, \dots, R$  do
5:       solve the eigenvalue problem  $\mathbb{M} \mathbf{w}_i = \lambda_i \mathbf{w}_i$ 
6:     end for
7:   else
8:      $\mathbb{K} = \mathbb{S} \mathbb{S}^T$ 
9:     for  $i = 1, \dots, R$  do
10:      solve the eigenvalue problem  $\mathbb{K} \mathbf{z}_i = \lambda_i \mathbf{z}_i$ 
11:       $\mathbf{w}_i = \frac{1}{\sqrt{\lambda_i}} \mathbb{S} \mathbf{z}_i$ 
12:    end for
13:   end if
14:   find the minimum  $L$  satisfying (2.83)
15:    $\mathbb{V} = [\mathbf{w}_1 \mid \dots \mid \mathbf{w}_L]$ 
16: end function

```

⁶Let $\mathbf{s} = \alpha_1 \mathbf{s}_1 + \dots + \alpha_N \mathbf{s}_N \in \text{Col}(\mathbb{V})$. Then:

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

2.5.2 Implementation: details and issues

The whole numerical procedure presented so far can be efficiently carried out within an offline-online framework [39]. The parameter-independent *offline* step consists of the generation of the snapshots through a high-fidelity, expensive discretization scheme (e.g., the finite element method) and the subsequent construction of the reduced basis through via POD. Then, given a new parameter value $\boldsymbol{\mu} \in \mathcal{P}$, the nonlinear reduced system (2.67) is solved *online* resorting to, e.g., the Newton's method, which entails the assembly and resolution of linear systems of the form (2.69). 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.

```

1: function  $[\mathbb{V}, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n] = \text{PODG\_OFFLINE}(\mathcal{P}, \Omega_h, N, \delta_{\text{POD}}, \delta_{\text{NWT}}, K_{\text{max}})$ 
2:   solve the Laplace problems (2.38) and (2.39), yielding  $\{\xi_i\}_{i=1}^n$  and  $\{\pi_i\}_{i=1}^n$ 
3:   generate the parameter set  $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$ 
4:    $\mathbf{u}_h(\boldsymbol{\mu}^{(i)}) = \text{NEWTON}(\boldsymbol{\mu}^{(i)}, \mathbf{u}_h^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\text{NWT}}, K_{\text{max}})$ , for  $i = 1, \dots, N$ 
5:    $\mathbb{S} = [\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \mid \dots \mid \mathbf{u}_h(\boldsymbol{\mu}^{(N)})]$ 
6:    $\mathbb{V} = \text{POD}(\mathbb{S}, \delta_{\text{POD}})$ 
7: end function

1: function  $\mathbf{u}_L(\boldsymbol{\mu}) = \text{PODG\_ONLINE}(\boldsymbol{\mu}, \Omega_h, \mathbb{V}, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\text{NWT}}, K_{\text{max}})$ 
2:    $\mathbf{u}_{\text{rb}}(\boldsymbol{\mu}) = \text{NEWTONRB}(\boldsymbol{\mu}, \Omega_h, \mathbb{V}, \mathbf{u}_{\text{rb}}^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\text{NWT}}, K_{\text{max}})$ 
3:    $\mathbf{u}_L(\boldsymbol{\mu}) = \mathbb{V} \mathbf{u}_{\text{rb}}(\boldsymbol{\mu})$ 
4: 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 (2.67) and (2.68) of the reduced residual vector $\mathbf{G}_{\text{rb}}(\cdot; \boldsymbol{\mu})$ and its Jacobian $\mathbb{J}_{\text{rb}}(\cdot; \boldsymbol{\mu})$, 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 $\Phi(\cdot; \boldsymbol{\mu})$), the assembly of the reduced linear systems (2.69) has to be embodied directly in the online stage, thus seriously compromising the efficiency of the overall procedure [3]. Without escaping the algebraic framework, this can be successfully overcome upon resorting to suitable techniques as the discrete empirical interpolation method (DEIM) or its matrix variant (MDEIM) [33]. The basic idea is to recover an affine dependency on the parameter $\boldsymbol{\mu}$, approximating the reduced residual vector in the form

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

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

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

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

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

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

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

and $\alpha_j^q(\cdot; \cdot)$, $q = 1, \dots, Q_j$, solution- and parameter-dependent scalar coefficients [41]. 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 $\alpha_j^q(\cdot; \cdot)$.

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 2.6 for a complete description and further motivations. Before that, as an instructive example, let us briefly discuss the application of the (standard) POD-Galerkin method to the steady incompressible Navier-Stokes equations.

2.5.3 Application to the steady Navier-Stokes equations

Let $\{(\mathbf{v}_h(\boldsymbol{\mu}^{(n)}), p_h(\boldsymbol{\mu}^{(n)}))\}_{n=1}^N \subset V_h$ be an ensemble of velocity and pressure FE snapshots, corresponding to the parameter values $\{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\} \subset \mathcal{P}$. Slightly distancing ourselves from the general workflow depicted in the previous sections, for the Navier-Stokes equations is convenient to construct two separate reduced spaces V_{rb}^v and V_{rb}^p for the velocity and pressure fields, respectively [2, 5, 7, 41]; the advantages of this approach will be clearer later in the section. In the light of this, let

$$\mathbb{S}_v = [\mathbf{v}_h(\boldsymbol{\mu}^{(1)}) \mid \dots \mid \mathbf{v}_h(\boldsymbol{\mu}^{(N)})] \in \mathbb{R}^{2M_n \times N} \quad \text{and} \quad \mathbb{S}_p = [\mathbf{p}_h(\boldsymbol{\mu}^{(1)}) \mid \dots \mid \mathbf{p}_h(\boldsymbol{\mu}^{(N)})] \in \mathbb{R}^{M_v \times N}$$

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

$$\mathbb{S}_v = \mathbb{W}_v \Sigma_v \mathbb{Z}_v^T \quad \text{and} \quad \mathbb{S}_p = \mathbb{W}_p \Sigma_p \mathbb{Z}_p^T,$$

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

$$\mathbb{V}_v = [\boldsymbol{\psi}_1^v \mid \dots \mid \boldsymbol{\psi}_{L_v}^v] \quad \text{and} \quad \mathbb{V}_p = [\boldsymbol{\psi}_1^p \mid \dots \mid \boldsymbol{\psi}_{L_p}^p],$$

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

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

with $\mathbf{v}_{\text{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^{L_v}$ and $\mathbf{p}_{\text{rb}}(\boldsymbol{\mu}) \in \mathbb{R}^{L_p}$. Plugging the expressions for $\mathbf{v}_L(\boldsymbol{\mu})$ and $\mathbf{p}_L(\boldsymbol{\mu})$ into the FE system (2.58) and enforcing the orthogonality with the columns of \mathbb{V}_v and \mathbb{V}_p , yields the following reduced-order nonlinear system for the $L = L_v + L_p$ RB coefficients gathered in $\mathbf{v}_{\text{rb}}(\boldsymbol{\mu})$ and $\mathbf{p}_{\text{rb}}(\boldsymbol{\mu})$:

$$\begin{bmatrix} \mathbb{V}_v^T & 0 \\ 0 & \mathbb{V}_p^T \end{bmatrix} \begin{bmatrix} \mathbb{A}(\boldsymbol{\mu}) + \mathbb{C}(\mathbb{V}_v \mathbf{v}_{\text{rb}}(\boldsymbol{\mu}); \boldsymbol{\mu}) & \mathbb{B}(\boldsymbol{\mu})^T \\ \mathbb{B}(\boldsymbol{\mu}) & 0 \end{bmatrix} \begin{bmatrix} \mathbb{V}_v \mathbf{v}_{\text{rb}}(\boldsymbol{\mu}) \\ \mathbb{V}_p \mathbf{p}_{\text{rb}}(\boldsymbol{\mu}) \end{bmatrix} = \begin{bmatrix} \mathbb{V}_v^T & 0 \\ 0 & \mathbb{V}_p^T \end{bmatrix} \begin{bmatrix} \mathbf{f}_1(\boldsymbol{\mu}) \\ \mathbf{f}_2(\boldsymbol{\mu}) \end{bmatrix}. \quad (2.84)$$

Developing the calculations and defining

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

the system (2.84) can be expressed in compact form as

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

Let us point out that the reduced system (2.84) can be recast in the canonical form (2.67) upon setting

$$\mathbb{V} = \begin{bmatrix} \mathbb{V}_v & 0 \\ 0 & \mathbb{V}_p \end{bmatrix} \in \mathbb{R}^{M \times L}.$$

Although in Section 2.4.2 the underpinning finite element solver has been designed to fulfill the inf-sup condition (2.45), thus ensuring the stability of the scheme, when dealing with incompressible flows this property could not be automatically inherited by the reduced-order solver, which as a result may show severe instabilities [5]. Indeed, one has to carefully chosen the reduced velocity and pressure spaces so to meet an equivalent of the inf-sup condition at the reduced level. To this end, in our simulations we resort to a *supremizer enrichment* of the reduced velocity space X_{rb} , as proposed in [2], to which we refer the readers which are interested in the theoretical rational behind this approach. In the following, we rather provide some insights on the supremizer enrichment, mainly focusing on algorithmic and implementation aspects.

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

$$\mathbb{M}_v \mathbf{s}(\boldsymbol{\mu}^{(n)}) = \mathbb{B}(\boldsymbol{\mu}^{(n)})^T \mathbf{p}_h(\boldsymbol{\mu}^{(n)}), \quad \text{for } n = 1, \dots, N. \quad (2.85)$$

where $\mathbb{B} = \mathbb{B}(\boldsymbol{\mu}) \in \mathbb{R}^{M_v \times 2M_n}$ has been introduced in Equation (2.57c), while $\mathbb{M}_v \in \mathbb{R}^{2M_n \times 2M_n}$ denotes the mass matrix in the $[H^1(\Omega)]^2$ scalar product for the velocity FE basis functions, and it is defined by

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

Letting $\mathbb{S}_s = [\mathbf{s}(\boldsymbol{\mu}^{(1)}) | \dots | \mathbf{s}(\boldsymbol{\mu}^{(N)})] \in \mathbb{R}^{2M_n \times N}$ collect the coefficients for the expansion of the supremizer solutions in the Langrangian FE basis, we generate a POD basis for the

supremizer space by performing the SVD of \mathbb{S}_s , and we then enrich the reduced basis for the velocity field by means of the first L_s left singular vectors $\{\boldsymbol{\psi}_i^s\}_{i=1}^{L_s}$, yielding an enriched reduced velocity space \tilde{X}_{rb} . In other terms, setting

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

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

$$\mathbf{v}_L(\boldsymbol{\mu}) = \bar{\mathbb{V}}_v \tilde{\mathbf{v}}_{rb}(\boldsymbol{\mu}) = \sum_{i=1}^{L_v} v_{rb}^{(i)} \boldsymbol{\psi}_i^v + \sum_{i=1}^{L_s} s_{rb}^{(i)} \boldsymbol{\psi}_i^s, \quad (2.86)$$

where

$$\bar{\mathbb{V}}_v = [\mathbb{V}_v \mid \mathbb{V}_s] \in \mathbb{R}^{2M_n \times (L_v + L_s)} \quad \text{and} \quad \tilde{\mathbf{v}}_{rb}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{v}_{rb}(\boldsymbol{\mu}) \\ \mathbf{s}_{rb}(\boldsymbol{\mu}) \end{bmatrix} \in \mathbb{R}^{L_v + L_s}.$$

Replacing \mathbb{V}_v with $\bar{\mathbb{V}}_v$ and \mathbf{v}_{rb} with $\tilde{\mathbf{v}}_{rb}$ into (2.84), we get the reduced nonlinear system for $(\tilde{\mathbf{v}}_{rb}(\boldsymbol{\mu}), \mathbf{p}_{rb}(\boldsymbol{\mu}))$, consisting of L_s additional equations. The stability of the resulting reduced-order scheme is ensured by the following proposition [2].

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

$$(\mathbf{w}_L, \mathbf{z}_L)_{rb} = \bar{\mathbf{w}}_{rb}^T \mathbb{M}_v^{rb} \bar{\mathbf{z}}_{rb} \quad \text{and} \quad (q_L, r_L) = \mathbf{q}_{rb}^T \mathbb{M}_p^{rb} \mathbf{r}_{rb},$$

where

$$\mathbb{M}_v^{rb} = \begin{bmatrix} \mathbb{V}_v^T \mathbb{M}_v \mathbb{V}_v & \mathbb{V}_v^T \mathbb{M}_v \mathbb{V}_s \\ \mathbb{V}_s^T \mathbb{M}_v \mathbb{V}_v & \mathbb{V}_s^T \mathbb{M}_v \mathbb{V}_s \end{bmatrix} \in \mathbb{R}^{(L_v + L_s) \times (L_v + L_s)} \quad \text{and} \quad \mathbb{M}_p^{rb} = \mathbb{V}_p^T \mathbb{M}_p \mathbb{V}_p \in \mathbb{R}^{L_p \times L_p},$$

with \mathbb{M}_v and \mathbb{M}_p being the mass matrices for the velocity and pressure (full-order) spaces, respectively. Assuming that the full-order model satisfies the inf-sup stability condition (2.55), the POD-Galerkin ROM with supremizer enrichment of the velocity space is inf-sup stable, that is

$$\beta_L(\boldsymbol{\mu}) = \inf_{q_L \in Q_{rb}} \sup_{\mathbf{w}_L \in \tilde{X}_{rb}} \frac{b_h(q_L, \mathbf{w}_L; \boldsymbol{\mu})}{\|\mathbf{w}_L\|_{rb} \|q_L\|_{rb}} \geq \beta_{0L} \geq \beta_0 > 0 \quad \forall \boldsymbol{\mu} \in \mathcal{P}. \quad (2.87)$$

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 and open questions for the standard POD-Galerkin reduced basis technique, particularly for nonlinear differential problems featuring a nonaffine dependence on the parameters. In those cases, we have already discussed in Section 2.5.2 about the incapability of the method to completely decouple the online stage from the underlying high-fidelity, expensive discrete model, thus preventing a 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 and then retain a larger number of POD modes to guarantee the same accuracy provided by the standard POD-Galerkin method [3].

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

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

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

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

We have seen that the above system encodes the projection of the original variational model onto the reduced space V_{rb} . For our purposes, it should worth remark that there exists a one-to-one correspondence between V_{rb} and $\text{Col}(\mathbb{V})$. 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 (2.81) follows:

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

In particular, this implies that the projection of any $v_h \in V_h$ onto V_{rb} in the discrete scalar product $(\cdot, \cdot)_h$ (see Equation (2.82)) algebraically corresponds to the projection of \mathbf{v}_h onto $\text{Col}(\mathbb{V})$ in the Euclidean scalar product, given by

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

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

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

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

$$\mathbf{u}_L \approx \mathbb{P} \mathbf{u}_h = \mathbb{V} \mathbb{V}^T \mathbf{u}_h, \quad (2.88a)$$

or equivalently

$$\mathbf{u}_{\text{rb}} \approx \mathbb{V}^T \mathbf{u}_h. \quad (2.88b)$$

However, the iterative solver, e.g., the Newton's method, applied to the reduced system (2.67) may provide a slow convergence to the root of the system, so that the computed solutions do not meet the optimality conditions (2.88). To get a sense of this, let us briefly consider the following boundary value problem for the nonlinear two-dimensional Poisson equation:

$$\begin{cases} -\tilde{\nabla} \cdot ((1 + \tilde{u}(\boldsymbol{\mu})) \tilde{\nabla} \tilde{u}(\boldsymbol{\mu})) = \tilde{s}(\tilde{x}, \tilde{y}) & \text{in } \tilde{\Omega}(\boldsymbol{\mu}), \\ \tilde{u}(\boldsymbol{\mu}) = \tilde{\sigma}_y \sin(\tilde{\sigma}_x) \cos(\tilde{\sigma}_y) & \text{on } \partial \tilde{\Omega}(\boldsymbol{\mu}). \end{cases} \quad (2.89)$$

where $\tilde{\Omega}(\boldsymbol{\mu})^7$ is the quadrilateral domain shown in Figure 2.3, with $\boldsymbol{\mu} = (\mu_1, \mu_2) \in [\pi/4, 3\pi/4] \times [0, \pi/2]$, and the source term \tilde{s} has been chosen so that the exact solution to the problem is

⁷Observe that here only geometrical parameters are involved, i.e., $\boldsymbol{\mu} \equiv \boldsymbol{\mu}_g$.

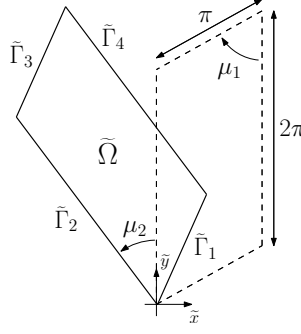


Figure 2.3. The parametrized computational domain (solid line) for the Poisson problem (2.89).

given by $\tilde{u}_{ex} = \tilde{y} \sin(\tilde{x}) \cos(\tilde{y})$. Assume the POD modes have been constructed relying on $N = 100$ parameter samples picked via the Latin Hypercube Sampling (LHS) [23], with the correspondent snapshots computed over a triangular mesh consisting of 1921 vertices. On the left of Figure 2.4, we compare the decay with L (i.e., the dimension of the reduced basis) for the relative errors between the high-fidelity solution and either its projection onto V_{rb} (red) or the Galerkin-POD RB approximation (blue). Given $\boldsymbol{\mu} \in \mathcal{P}$, the errors are respectively defined as

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

In particular, the plot reports the average values $\bar{\varepsilon}_V(L)$ and $\bar{\varepsilon}_{\text{PODG}}(L)$ of $\varepsilon_V(L, \boldsymbol{\mu})$ and $\varepsilon_{\text{PODG}}(L, \boldsymbol{\mu})$, respectively, evaluated over a test parameter set $\Xi_{te} \subset \mathcal{P}$ consisting of $N_{te} = 50$ samples, with $\Xi_{te} \cap \Xi_N = \emptyset$. We observe that although generally the Galerkin-POD scheme shows an exponential convergence to the high-fidelity solution, attaining a relative error of 0.1% for $L = 50$, for $L > 25$ it may yield an error which differs up to one order of magnitude from the lower-bound provided by the projection error. Since this latter decreases monotonically, the cause has to be sought in the Newton's method. In addition, even when the solver successfully converges to the solution, resorting to a reduced-order strategy only seldomly enables a remarkable computational saving with respect to the finite element method, as shown by the plot on the right of Figure 2.4, reporting the online run times on each test sample for both the full-order and the reduced-order scheme⁸. As already pointed out before, this has to be ascribed to the nonaffine parametric dependence featured by the problem.

The scenario portrayed so far motivates the research for an alternative approach to tackle any online query within the reduced basis framework, hopefully skipping the assembly and resolution of the reduced system. To this end, we have noted above that the element of V_{rb} closest to the high-fidelity solution u_h in the discrete norm $\|\cdot\|_h = \sqrt{(\cdot, \cdot)_h}$ can be expressed as

$$u_h^V(\mathbf{x}; \boldsymbol{\mu}) = \sum_{j=1}^M (\mathbb{V}\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}))_j \phi_j(\mathbf{x}) = \sum_{i=1}^L (\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}))_i \psi_i(\mathbf{x}). \quad (2.90)$$

Motivated by last equality, once a reduced basis has been constructed (e.g., via POD of the

⁸In this respect, we mention that all the results presented throughout this thesis have been obtained on a Lenovo laptop equipped with a CPU Intel Core i7 @ 2.50 GHz.

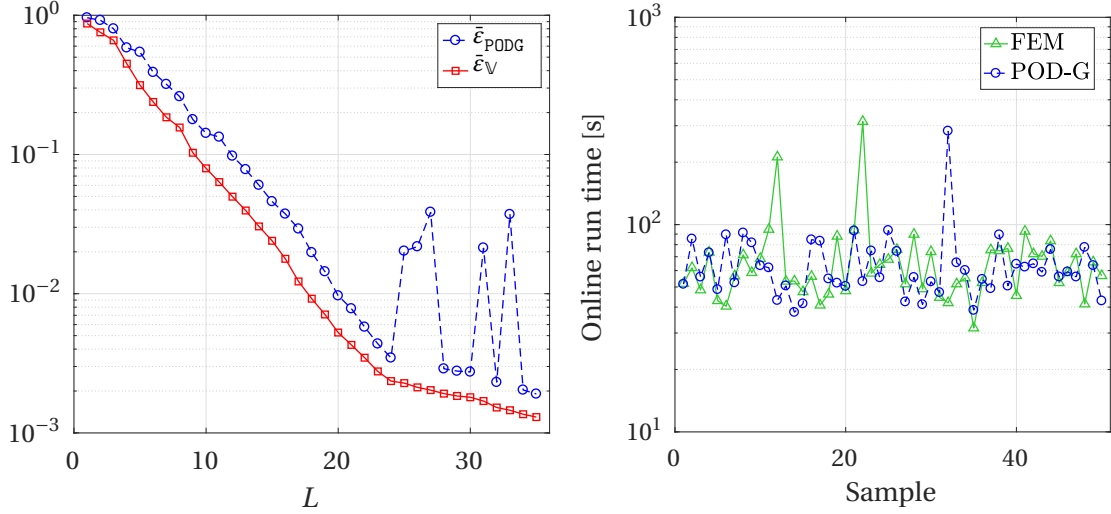


Figure 2.4. Left: average relative error between the FE solution for the problem (2.89) and either its projection onto the reduced space (red) or the POD-Galerkin RB solution (blue); the errors have been evaluated on a test parameter set $\Xi_{te} \subset \mathcal{P}$ consisting of $N_{te} = 50$ randomly picked values. Right: comparison between the online run times for the FE (full-order) scheme and the POD-Galerkin (reduced-order) method on Ξ_{te} .

snapshot matrix), we aim at approximating the function

$$\begin{aligned} \boldsymbol{\pi} : \mathcal{P} \subset \mathbb{R}^P &\rightarrow \mathbb{R}^L \\ \boldsymbol{\mu} &\mapsto \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}), \end{aligned} \quad (2.91)$$

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

$$\mathbf{u}_{rb}(\boldsymbol{\mu}) = \hat{\boldsymbol{\pi}}(\boldsymbol{\mu}), \quad (2.92)$$

and, consequently,

$$\mathbf{u}_L(\boldsymbol{\mu}) = \mathbb{V} \hat{\boldsymbol{\pi}}(\boldsymbol{\mu}). \quad (2.93)$$

It is worth pointing 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 these facts in the next section.

In the literature, different approaches for the *interpolation* of (2.91) have been developed, e.g., exploiting some geometrical considerations concerning the solution manifold \mathcal{M} [1], or employing radial basis functions [7]. 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, training the perceptron via exposition to a collection of (known) input-output pairs

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

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 $\mathbf{t}_i = \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}^{(i)}) \in \mathbb{R}^L$ the associated *teaching input*; together, they constitute a *training pattern*. In this respect, note that the teaching inputs $\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}^{(i)})$, $i = 1, \dots, N_{tr}$, are generated through the FE solver. On the one hand, this ensures the reliability of the teaching patterns, given the assumed high-fidelity of the FE scheme (conversely to the reduced solver, as already appreciated before). On the other hand, this also highly suggests to incorporate the learning phase of the perceptron within the offline step of the POD-NN RB method, as described in Algorithm 2.6. In doing so, we exploit the natural decoupling between the training and the evaluation of neural networks, thus fulfilling the necessary requirement to enable great online efficiency; we refer the reader to the following chapter for a numerical validation of this assertion.

However, let us point out that the design of an effective learning procedure may require a larger amount of snapshots than the generation of the reduced space does. Moreover, we have extensively discussed in Chapter 1 about the time-consuming yet unavoidable *trial-and-error* approach which one should pursue in the research for an optimal network topology. To this end, we propose an automatic procedure, resumed in Algorithm 2.5.

While the Cybenko's theorems (see (i) and (ii) in Section 1.2.2) let us confine ourselves to perceptrons with no more than two hidden layers, no similar a priori and general results are available for the number H of neurons per layer (to ease the work, we uniquely consider networks with the same number of neurons in both the first and the second hidden layer). Hence, given an initial amount N_{tr} of training samples (say $N_{tr} = 100$), we train the network for increasing values of H , stopping when overfitting of training data occurs (due to an excessive number of neurons with respect to the number of training patterns). In case the best configuration, i.e., the one yielding the smallest error on a test data set, does not meet a desired level of accuracy, we generate a new set of snapshots, which will enlarge the current training set, and we then proceed to re-train the networks. At this point, it worths to point out two remarks.

- (i) We can now limit ourselves to network configurations including a number of neurons non-smaller than the current optimal network. Indeed, the error (on the test data set) yielded by a neural network is reasonably decreasing in the number of patterns it is exposed to during the learning phase, and the larger the number of neurons, the faster the decay.
- (ii) In order to maximize the additional quantity of information available for the learning, we should ensure that the new training inputs, i.e., parameter values, do not overlap with the ones already present in the training parameter set. To this aim, we pursue an euristic approach, employing, at each iteration, the latin hypercube sampling, which has proved to provide a good compromise between randomness and even coverage of the parameter domain; an example is offered in Figure 2.5 for $\mathcal{P} \subset \mathbb{R}^2$.

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

Algorithm 2.5 Selection of an optimal network configuration.

```

1: function ( $\mathcal{N}^{opt}, \mathcal{V}^{opt}, \mathbf{w}^{opt}$ ) =
2:   PODNN_TRAINING( $\mathbb{V}, \varepsilon, N_{tr}^{max}, H^{max}, N_{te}, d, K_{ms}, \delta, T, K_{ea}$ )
3:   set  $\Delta N = 100, H_0 = 5, \Delta H = 5, N_{va} = \lceil 0.3 N_{tr}^{max} \rceil$   $\triangleright$  Adjustable parameters
4:   randomly drawn validation and test samples  $\Xi_{va} = \{\boldsymbol{\mu}_{va}^{(i)}\}_{i=1}^{N_{va}}$  and  $\Xi_{te} = \{\boldsymbol{\mu}_{te}^{(i)}\}_{i=1}^{N_{te}}$ 
5:   compute  $\{\mathbf{u}_h(\boldsymbol{\mu}_{va}^{(i)})\}_{i=1}^{N_{va}}$  and  $\{\mathbf{u}_h(\boldsymbol{\mu}_{te}^{(i)})\}_{i=1}^{N_{te}}$ 
6:   set  $P_{va} = \{(\boldsymbol{\mu}_{va}^{(i)}, \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}_{va}^{(i)}))\}_{i=1}^{N_{va}}$  and  $P_{te} = \{(\boldsymbol{\mu}_{te}^{(i)}, \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}_{te}^{(i)}))\}_{i=1}^{N_{te}}$ 
7:   evaluate projection error  $E_{\text{POD}} = 1/N_{te} \sum_{i=1}^{N_{te}} \|\mathbf{u}_h(\boldsymbol{\mu}_{te}^{(i)}) - \mathbb{V} \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}_{te}^{(i)})\|$ 
8:    $P_{tr} = \emptyset, N_{tr} = 0, H^{opt} = H_0$ 
9:   do
10:     $N_{tr} \leftarrow N_{tr} + \Delta N$ 
11:    generate training samples  $\Xi_{tr} = \{\boldsymbol{\mu}_{tr}^{(i)}\}_{i=1}^{N_{tr}}$  via Latin Hypercube Sampling (LHS)
12:    compute snapshots  $\{\mathbf{u}_h(\boldsymbol{\mu}_{tr}^{(i)})\}_{i=1}^{N_{tr}}$ 
13:    extend training data set:  $P_{tr} \leftarrow P_{tr} \cup \{(\boldsymbol{\mu}_{tr}^{(i)}, \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}_{tr}^{(i)}))\}_{i=1}^{N_{tr}}$ 
14:     $E_{\text{NN}}^{opt} = \infty, H = H^{opt}$ 
15:    while  $H \leq H^{max}$  and  $E_{te} < E_{te}^{old}$  do
16:      design a three-layers perceptron  $(\mathcal{N}, \mathcal{V})$  with  $H$  neurons per hidden layer
17:       $[\mathbf{w}, E_{\text{NN}}] = \text{TRAINING}((\mathcal{N}, \mathcal{V}), P_{tr}, P_{va}, P_{te}, d, K_{ms}, \delta, T, K_{ea})$ 
18:      if  $E_{\text{NN}} < E_{\text{NN}}^{opt}$  then
19:         $E_{\text{NN}}^{opt} = E_{\text{NN}}, \mathcal{N}^{opt} = \mathcal{N}, \mathcal{V}^{opt} = \mathcal{V}, \mathbf{w}^{opt} = \mathbf{w}, H^{opt} = H$ 
20:         $H \leftarrow H + \Delta H$ 
21:      else  $\triangleright$  Too many neurons: the network overfits training data
22:        break
23:      end if
24:    end while
25:    while  $N_{tr} \leq N_{tr}^{max}$  and  $|E_{\text{POD}} - E_{\text{NN}}^{opt}| > \varepsilon$ 
26:  end function

```

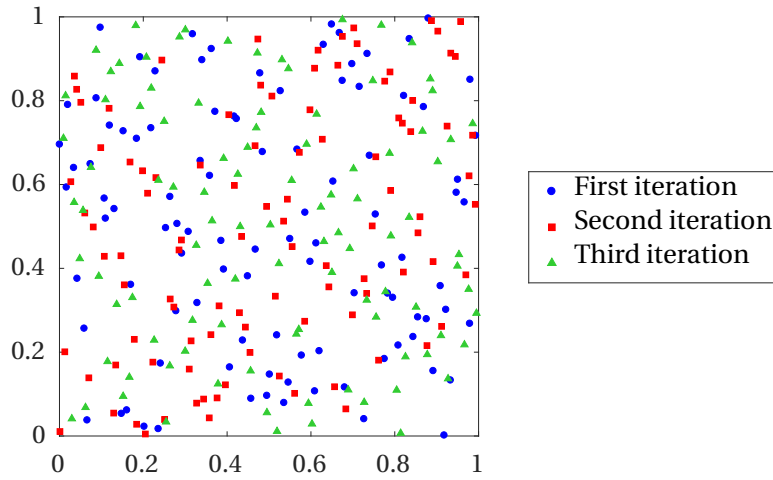


Figure 2.5. Three point sets randomly generate through the latin hypercube sampling. Observe the good coverage provided by the union of the sets, with only a few overlapping points.

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

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

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

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

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

$$\begin{aligned} \text{MSE}(\mathbf{u}_{\text{rb}}^{\text{NN}}, \mathbb{V}^T \mathbf{u}_h) &\propto \|\mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V}^T \mathbf{u}_h\|_{\mathbb{R}^L}^2 = (\mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V}^T \mathbf{u}_h)^T (\mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V}^T \mathbf{u}_h) \\ &= (\mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V}^T \mathbf{u}_h)^T \underbrace{\mathbb{V}^T \mathbb{V}}_{\mathbb{I}_L \in \mathbb{R}^{L \times L}} (\mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V}^T \mathbf{u}_h) \\ &= (\mathbb{V} \mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V} \mathbb{V}^T \mathbf{u}_h)^T (\mathbb{V} \mathbf{u}_{\text{rb}}^{\text{NN}} - \mathbb{V} \mathbb{V}^T \mathbf{u}_h) \\ &= \|\mathbf{u}_L^{\text{NN}} - \mathbb{V} \mathbb{V}^T \mathbf{u}_h\|_{\mathbb{R}^M}^2 = \|u_L^{\text{NN}} - u_h^{\mathbb{V}}\|_h^2, \end{aligned} \quad (2.94)$$

where

$$u_L^{\text{NN}}(\mathbf{x}; \boldsymbol{\mu}) = \sum_{i=1}^L (\mathbf{u}_{\text{rb}}^{\text{NN}}(\boldsymbol{\mu}))_i \psi_i(\mathbf{x}) \in V_{\text{rb}}. \quad (2.95)$$

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_{rb} for all the training inputs $\boldsymbol{\mu}^{(i)}$, $i = 1, \dots, N_{\text{tr}}$. 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 2.6 The offline and online stages for the POD-NN RB method.

- 1: **function** $[\mathbb{V}, \mathcal{N}_{\text{rb}}, \mathcal{V}_{\text{rb}}, \mathbf{w}_{\text{rb}}] = \text{PODNN_OFFLINE}(\mathcal{P}, \Omega_h, N, \delta_{\text{NWT}}, K_{\text{max}}, \delta_{\text{POD}}, \delta_{\text{NN}}, \varepsilon, N_{\text{tr}}^{\text{max}}, H^{\text{max}}, N_{\text{te}}, d, K_{\text{ms}}, T, K_{\text{ea}})$
 - 2: solve the Laplace problems (2.38) and (2.39), yielding $\{\xi_i\}_{i=1}^n$ and $\{\pi_i\}_{i=1}^n$
 - 3: generate the parameter set $\Xi_N = \{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(N)}\}$
 - 4: $\mathbf{u}_h(\boldsymbol{\mu}^{(i)}) = \text{NEWTON}(\boldsymbol{\mu}^{(i)}, \mathbf{u}_h^0, \{\xi_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \delta_{\text{NWT}}, K_{\text{max}})$, for $i = 1, \dots, N$
 - 5: $\mathbb{S} = [\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \mid \dots \mid \mathbf{u}_h(\boldsymbol{\mu}^{(N)})]$
 - 6: $\mathbb{V} = \text{POD}(\mathbb{S}, \delta_{\text{POD}})$
 - 7: $(\mathcal{N}_{\text{rb}}, \mathcal{V}_{\text{rb}}, \mathbf{w}_{\text{rb}}) = \text{PODNN_TRAINING}(\mathbb{V}, \varepsilon, N_{\text{tr}}^{\text{max}}, H^{\text{max}}, N_{\text{te}}, d, K_{\text{ms}}, \delta_{\text{NN}}, T, K_{\text{ea}})$
 - 8: **end function**
-
- 1: **function** $\mathbf{u}_L^{\text{NN}}(\boldsymbol{\mu}) = \text{PODNN_ONLINE}(\boldsymbol{\mu}, \mathbb{V}, \mathcal{N}_{\text{rb}}, \mathcal{V}_{\text{rb}}, \mathbf{w}_{\text{rb}})$
 - 2: evaluate the output $\mathbf{u}_{\text{rb}}^{\text{NN}}(\boldsymbol{\mu})$ of the network $(\mathcal{N}_{\text{rb}}, \mathcal{V}_{\text{rb}}, \mathbf{w}_{\text{rb}})$ for the input vector $\boldsymbol{\mu}$
 - 3: $\mathbf{u}_L^{\text{NN}}(\boldsymbol{\mu}) = \mathbb{V} \mathbf{u}_{\text{rb}}^{\text{NN}}(\boldsymbol{\mu})$
 - 4: **end function**
-

While we shall numerically prove the actual effectiveness and efficiency of the POD-NN method in the upcoming chapter, in the following concluding section we aim at further

investigating the accuracy which the proposed reduced basis strategy could provide. To this end, we develop a simplified *a priori* error analysis. Yet, no rigorous proof is provided; rather, the goal is to give some insights on the 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 $u_h^\vee(\boldsymbol{\mu}) \in V_{\text{rb}}$ its projection onto V_{rb} , and $u_L^{\text{NN}}(\boldsymbol{\mu}) \in V_{\text{rb}}$ the reduced solution provided by the POD-NN method, defined in Equation (2.95). 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:

$$\|u - u_L^{\text{NN}}\|_{L^2(\Omega)} \leq \|u - u_h\|_{L^2(\Omega)} + \|u_h - u_h^\vee\|_{L^2(\Omega)} + \|u_h^\vee - u_L^{\text{NN}}\|_{L^2(\Omega)}. \quad (2.96)$$

Let us analyze the three terms appearing on the right-hand side of (2.96). 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:

$$\|u - u_h\|_{L^2(\Omega)} \leq \|u - u_h\|_V \leq \delta_{\text{HF}}, \quad (2.97)$$

with $\delta_{\text{HF}} > 0$ a given tolerance. Then, the term $\|u_h - u_h^\vee\|_{L^2(\Omega)}$ measures the distance between the truth solution and the reduced space V_{rb} . From the definitions (2.47) and (2.90), it follows that:

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

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:

$$\|u_h - u_h^\vee\|_{L^2(\Omega)}^2 = \|\mathbb{M}^{1/2}(\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h)\|_{\mathbb{R}^M}^2 \leq \|\mathbb{M}\|_2 \|\mathbf{u}_h - \mathbb{V}\mathbb{V}^T \mathbf{u}_h\|_{\mathbb{R}^M}^2, \quad (2.98)$$

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

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

i.e., the distance (in the Euclidean norm) between $\mathbf{u}_h \in \mathbb{R}^M$ and its projection onto the column space $\text{Col}\mathbb{V}$ 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 $\mathbb{S} = [\mathbf{u}_h(\boldsymbol{\mu}^{(1)}) \mid \dots \mid \mathbf{u}_h(\boldsymbol{\mu}^{(N)})] \in \mathbb{R}^{M \times N}$. From (2.79), 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:

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

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 \mathcal{M}_{Ξ_N} . Whereas, to be extendible to the entire discrete solution manifold \mathcal{M}_h , we have to assume that \mathcal{M}_h is a low-dimensional subspace of V_h , so that the ensemble of snapshots is actually representative of the entire \mathcal{M}_h . Then, further assuming a rapid (i.e., exponential) decay of the singular values with the order, we have

$$\|u_h - u_h^\mathbb{V}\|_{L^2(\Omega)}^2 \approx \beta e^{-\alpha L}. \quad (2.99)$$

The last term involved in (2.96) regards the distance between $u_h^\mathbb{V}$ and u_L^{NN} , i.e. the POD-NN solution. Similar to what done for the previous term, recalling the definition (2.95) for u_L^{NN} we derive:

$$\begin{aligned} \|u_h^\mathbb{V} - u_L^{\text{NN}}\|_{L^2(\Omega)}^2 &= \int_{\Omega} |u_h^\mathbb{V} - u_L^{\text{NN}}|^2 d\Omega = \int_{\Omega} \left| \sum_{i=1}^M (\mathbb{V}\mathbb{V}^T \mathbf{u}_h)_i \phi_i - \sum_{i=1}^M (\mathbb{V}\mathbf{u}_{\text{rb}}^{\text{NN}})_i \phi_i \right|^2 d\Omega \\ &= \sum_{i=1}^M \sum_{j=1}^M (\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\text{rb}}^{\text{NN}})_i (\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\text{rb}}^{\text{NN}})_j \underbrace{\int_{\Omega} \phi_i \phi_j d\Omega}_{\mathbb{M}_{i,j}} \\ &= (\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\text{rb}}^{\text{NN}})^T \mathbb{M} (\mathbb{V}\mathbb{V}^T \mathbf{u}_h - \mathbb{V}\mathbf{u}_{\text{rb}}^{\text{NN}}) \\ &= \|\mathbb{M}^{1/2} \mathbb{V} (\mathbb{V}^T \mathbf{u}_h - \mathbf{u}_{\text{rb}}^{\text{NN}})\|_{\mathbb{R}^M}^2 \\ &\leq \|\mathbb{M}\|_2 \|\mathbb{V}\|_2^2 \|\mathbb{V}^T \mathbf{u}_h - \mathbf{u}_{\text{rb}}^{\text{NN}}\|_{\mathbb{R}^L}^2 \leq L \|\mathbb{M}\|_2 \|\mathbb{V}^T \mathbf{u}_h - \mathbf{u}_{\text{rb}}^{\text{NN}}\|_{\mathbb{R}^L}^2, \end{aligned} \quad (2.100)$$

where the last inequality follows from

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

with $\|\cdot\|_F$ the Frobenius norm and $\mathbb{I}_L \in \mathbb{R}^{L \times L}$ the identity matrix of dimension L . In (2.100), by Equation (2.94) the term

$$\|\mathbb{V}^T \mathbf{u}_h - \mathbf{u}_L^{\text{NN}}\|_{\mathbb{R}^L} \quad (2.101)$$

coincides with the specific error function used to train the network. Then, as for (2.97), (2.101) can be lowered to any given tolerance δ_{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 (2.91) 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 (2.101) 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 (2.97), (2.99) and (2.100) into (2.96), we obtain the following estimate for the POD-NN error in the $L^2(\Omega)$ -norm:

$$\|u - u_L^{\text{NN}}\|_{L^2(\Omega)} \leq \delta_{\text{HF}} + \beta e^{-\alpha L} + \gamma \sqrt{L} \delta_{\text{NN}}, \quad (2.102)$$

with α , β and γ positive constants, independent of both the solution and the size of the reduced basis, and δ_{HF} and δ_{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 (2.91).

Chapter 3

Numerical results

In this chapter, we present different numerical results obtained via the POD-G and the POD-NN reduced basis (RB) methods applied to parametrized full-Dirichlet boundary value problems (BVPs) for the (nonlinear) Poisson equation and the incompressible steady Navier-Stokes equations. The problems which shall be treated involve from 1 up to 3 parameters, either physical or geometrical. For the Poisson equation, we consider both linear and nonlinear problems, stated on either one- or two-dimensional domains. In the monodimensional case, we deal uniquely with physical parameters, addressing the diffusion coefficient, the source term or the boundary conditions; whereas, in two spatial dimensions we let the parameters only affect the shape of the domain, not the state equation. This latter remark also holds for the test cases concerning the Navier-Stokes equations. Particularly, we shall tackle the well-known lid-driven cavity problem for different values of the Reynolds number. The application of both the POD-G and the POD-NN techniques to differential problems featuring both physical and geometrical parameters, as well as natural or periodic boundary conditions, is left as future work.

The two RB methodologies considered in this work are compared both in terms of accuracy and performance provided at the online stage. To this end, let $\mathbf{u}_L(\boldsymbol{\mu})$ and $\mathbf{u}_L^{\text{NN}}(\boldsymbol{\mu})$ be respectively the (algebraic) POD-G and POD-NN solutions of reduced order L to the general $\boldsymbol{\mu}$ -dependent problem (2.21). Furthermore, let $\|\cdot\|$ be the canonical (Euclidean) norm on \mathbb{R}^M . Once the offline phases for both the POD-G and the POD-NN method have been carried out, given a new parameter value $\boldsymbol{\mu} \in \mathcal{P}$ (not involved either in the generation of the POD basis or in the detection of an optimal neural network) the following *relative* errors with respect to the high-fidelity (finite element) solution $\mathbf{u}_h(\boldsymbol{\mu})$ are analyzed:

- (a) the POD-G relative error, that is

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

- (b) the POD-NN relative error, that is

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

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

(2.82)) onto the reduced space V_{rb} , namely

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

recall that (3.3) provides a lower-bound for both (3.1) and (3.2).

The above errors are then evaluated on a test parameter set $\Xi_{te} \subset \mathcal{P}$ consisting of N_{te} randomly picked samples. Upon choosing a sufficiently large number of test values, statistics for $\varepsilon_{\text{PODG}}(L, \cdot)$, $\varepsilon_{\text{PODNN}}(L, \cdot)$ and $\varepsilon_{\mathbb{V}}(L, \cdot)$ on Ξ_{te} can be reasonably assumed independent of the particular choice for Ξ_{te} , thus making any subsequent error analysis reliable. In particular, in our numerical studies we let N_{te} ranging from 50 up to 200, and we then consider the average, respectively denoted by

$$\bar{\varepsilon}_{\text{PODG}} = \bar{\varepsilon}_{\text{PODG}}(L), \quad \bar{\varepsilon}_{\text{PODNN}} = \bar{\varepsilon}_{\text{PODNN}}(L) \quad \text{and} \quad \bar{\varepsilon}_{\mathbb{V}} = \bar{\varepsilon}_{\mathbb{V}}(L)$$

and occasionally the maximum, respectively indicated as

$$\varepsilon_{\text{PODG}}^{\max} = \varepsilon_{\text{PODG}}^{\max}(L), \quad \varepsilon_{\text{PODNN}}^{\max} = \varepsilon_{\text{PODNN}}^{\max}(L) \quad \text{and} \quad \varepsilon_{\mathbb{V}}^{\max} = \varepsilon_{\mathbb{V}}^{\max}(L),$$

of the errors over the test data set.

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

A last relevant remark concerns the choice of the number of POD modes L to be retained in the model. To this end, in a production framework one could employ, e.g., the relative information content criterium (2.83). Here, for the sake of analysis, we pursue a different approach, evaluating (3.1), (3.2) and (3.3) (averaged over Ξ_{te}) for some (and often all) $L \leq L_{\max}$ (say $L_{\max} = 50$). In this respect, we point out that within the POD-NN framework, once a neural network has been (properly) trained for the approximation of the map (2.91) for a given L , then the approximations of the very same map for any $\bar{L} < L$ comes freely - simply retain the first \bar{L} components of the network output vector. Therefore, no additional training is required. Indeed, decorating the matrix \mathbb{V} with a superscript denoting the dimension of the represented basis, one has

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

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

3.1 One-dimensional Poisson equation

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

3.1.1 Linear test case

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

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

with $\sigma = 0.2$ (fixed) and $\mu \in [-1, 1]$. Here, the quote denotes a single derivative with respect to the spatial variable. The equation is completed with homogeneous Dirichlet boundary conditions. In order to derive a POD-Galerkin reduced basis model for (3.4), we first generate a collection of linear finite elements discretizations $\{\mathbf{u}_h(\mu^{(1)}), \dots, \mathbf{u}_h(\mu^{(N)})\}$ over a (physical) uniform grid with 100 points. Here, the latin hypercube sampling is not usable, since the parameter domain $\mathcal{P} \equiv [-1, 1]$ is monodimensional, hence we displace the training samples $\{\mu^{(1)}, \dots, \mu^{(N)}\}$ uniformly over \mathcal{P} . In the left plot of Figure 3.1, we analyze the relative error between FEM and the resulting Galerkin-POD method, averaged over an ensemble Ξ_{te} of $N_{te} = 100$ parameter values. Independently of the number of snapshots used to generate the POD basis, we can appreciate an exponential decay of the error with the number L of POD modes retained by the model, coherently with the behaviour exhibited by the singular values. The effectiveness of the RB approximation is confirmed by the other plot of Figure 3.1, showing the FE (solid line) and the POD-G (circles) solutions for three values of μ , with $N = 50$ and $L = 10$.

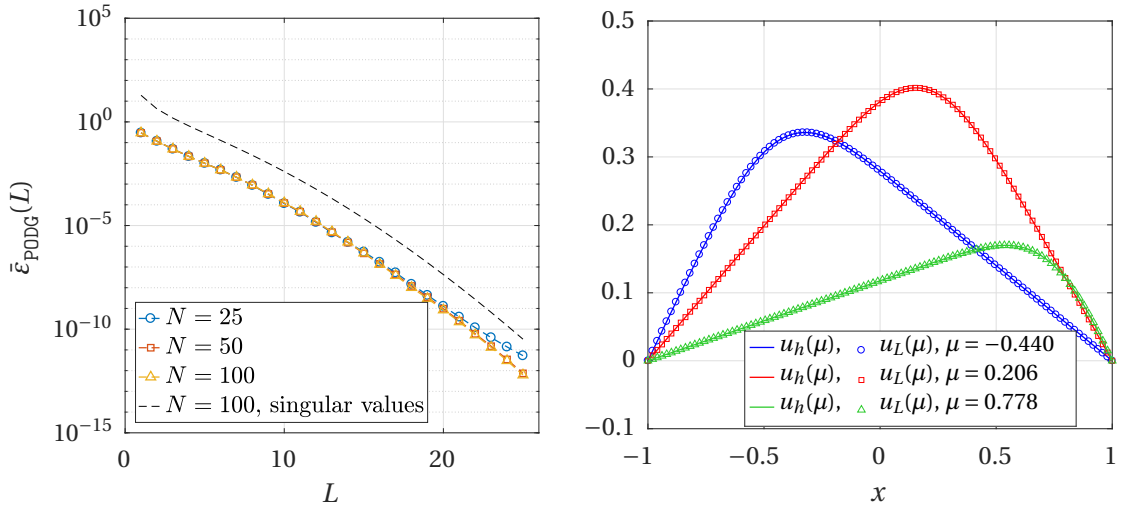


Figure 3.1. Left: error analysis for the POD-G RB method applied to Equation (3.4); the results refer to a POD basis constructed based on $N = 25, 50, 100$ snapshots; for $N = 100$, the singular values of the correspondent snapshot matrix are shown as well. Right: FE and POD-G solutions for three parameter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

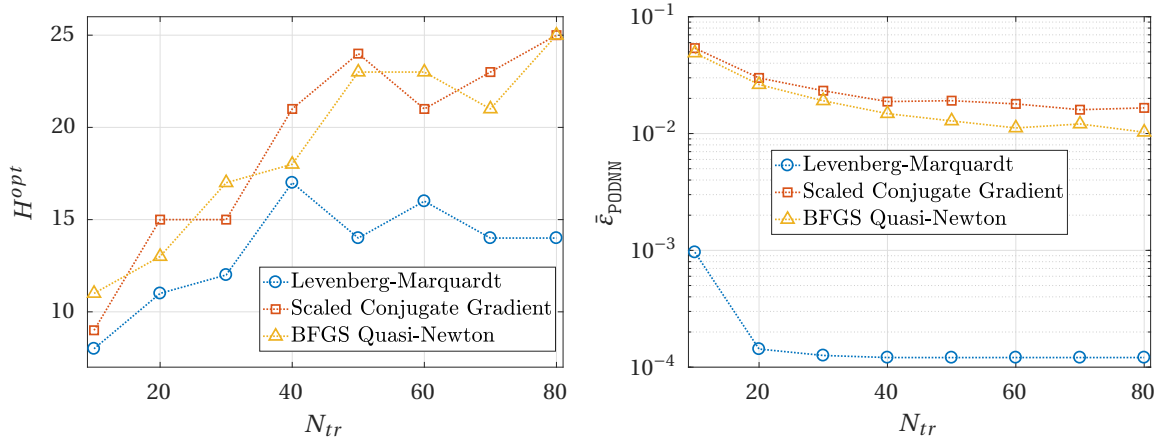


Figure 3.2. In respect to the approximation of the map (2.91) for the Poisson problem (3.4) via neural networks, comparison between three supervised algorithms - Levenberg-Marquardt (blue), Scaled Conjugate Gradient (orange), BFGS Quasi-Newton (yellow) - in terms of the optimal configuration detected (*right*) and the associated test error (*right*) for different amounts of training samples.

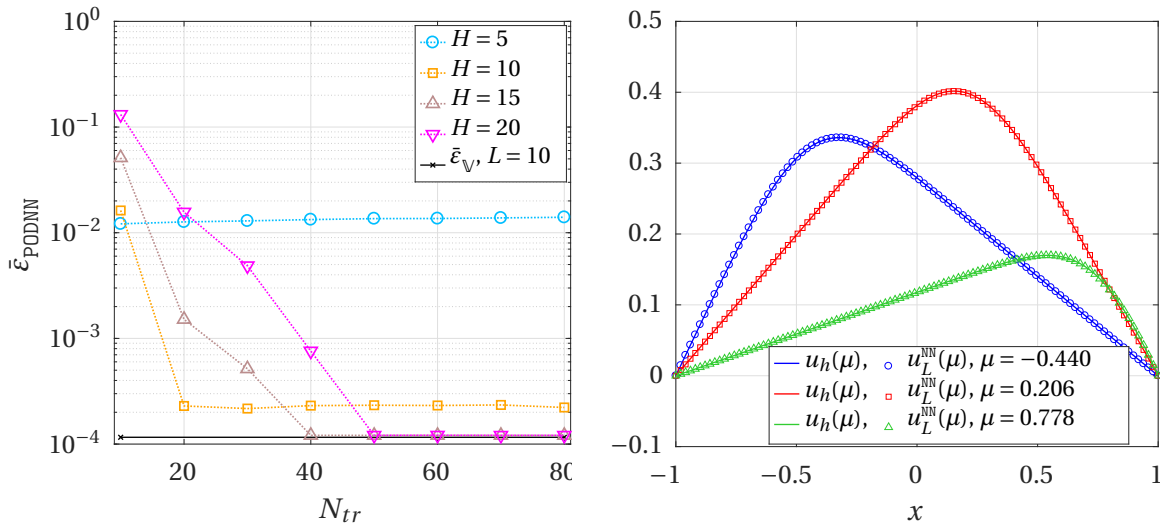


Figure 3.3. Left: relative error yielded by the POD-NN RB method applied to Equation (3.4); the convergence to the projection error for $L = 10$ (solid line) is analyzed in terms of both the number of neurons included in the neural network and the dimension of the training set. Right: FE and POD-NN solutions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

Moving to the analysis of the POD-NN method, we first seek to motivate the employment throughout this work of the Levenberg-Marquardt learning algorithm (see Section 1.2.3). To this end, we compare it against two other well-known and widely used supervised training routines - the Scaled Conjugate Gradient [34] and the BFGS Quasi-Newton method [13] - on the regression of the map $\pi = \pi(\mu) : \mu \mapsto \mathbb{V}^T \mathbf{u}_h(\mu)$ (see also Equation (2.91)), with $L = 10$ POD modes, i.e., $\mathbb{V} \in \mathbb{R}^{M \times 10}$. In this simple linear case, we can limit ourselves to networks with a single hidden layer. Then, for a given number N_{tr} of training samples, for each algorithm Figure 3.2 tracks the number of hidden neurons required to yield the minimum

error on Ξ_{te} (*left*) and the optimal error itself (*right*). We can observe that for any tested value of N_{tr} the Levenberg-Marquardt algorithm (blue line) leads to much more accurate approximations using fewer neurons, that is, it turns out to be not only effective but even efficient. A confirmation of this is given by the plot on the left in Figure 3.3, showing the decay of the POD-NN error as the number of employed patterns and neurons increases. Here, the lower-bound given by the projection error is actually attained via, e.g., a network with $H = 15$ hidden neurons, exposed to $N_{tr} = 40$ training samples. The solutions obtained by means of such a network are provided on the right plot of Figure 3.3; note the substantial overlap with the correspondent FE solutions.

3.1.2 Nonlinear test case, two parameters

The first nonlinear problem we consider reads:

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

where $\boldsymbol{\mu} = (\mu_1, \mu_2) \in \mathcal{P} = [1, 3] \times [1, 3]$ and the source term $s(x; \boldsymbol{\mu})$ is chosen so that $u_{ex}(x; \boldsymbol{\mu}) = \mu_2 \sin(2 + \mu_1 x)$ represents the exact solution to the problem for any $\boldsymbol{\mu} \in \mathcal{P}$. Here, a POD-Galerkin RB model built upon $N = 50$ samples (randomly picked via LHS) turns out to be highly effective. Indeed, as reported by Figure 3.4, the yielded relative error, averaged over $N_{te} = 100$ samples, rapidly decays as the amount L of retained POD basis functions increases, attaining an error of 10^{-7} already for $L = 15$. Hence, the design of an effective neural network, which can at least approach the accuracy provided by the algebraic ROM, is really challenging. The results concerning the research of an optimal three-layers network configuration via the methodology described in Algorithm 2.5 are provided on the left in Figure 3.5. The steps pursued by the routine are represented through solid tracts; however, for the sake of completeness we also report the test error (understood as the minimum over multiple restarts) for any considered value for both the number of hidden neurons per layer (H_1 and H_2) and the dimension of the training set (N_{tr}).

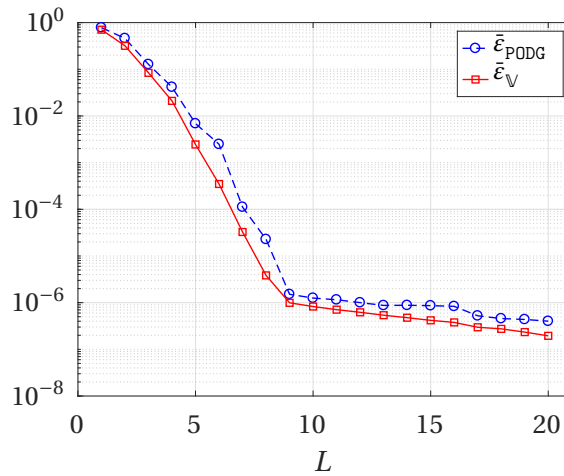


Figure 3.4. Error analysis for the POD-Galerkin RB method applied to the problem (3.5). The lower-bound provided by the projection error (3.3) is reported as reference.

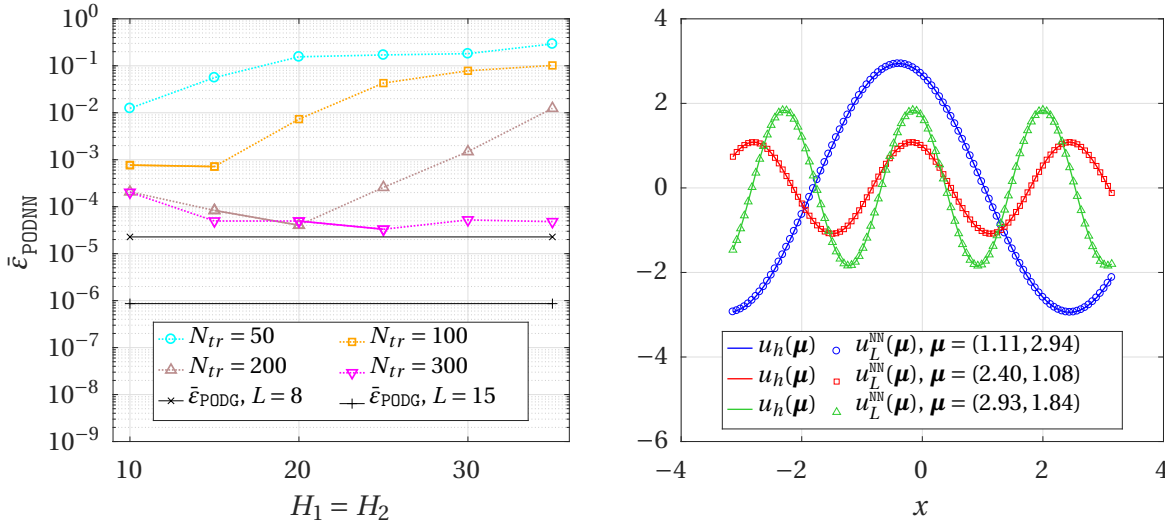


Figure 3.5. Left: error analysis for the POD-NN RB method (dotted lines) applied to problem (3.5). The solid solid sections refer to the steps carried out by the routine 2.5; the error yielded by the POD-G method using $L = 8$ or $L = 15$ basis functions are reported as reference. Right: comparison between FE and POD-NN solutions for three input vectors.

Let us first point out that the basic assumption which Algorithm 2.5 relies on is here fulfilled: as the number of available samples increase, the amount of neurons which allows to attain the minimum of the error increases as well, while the minimum itself decreases. However, although the networks have been trained to approximate the function (2.91) with $L = 15$ POD modes, we can not guarantee the same precision provided by the POD-G method retaining the same quantity of basis functions. Indeed, we get the same level of accuracy offered by POD-G with only $L = 8$ basis vectors. To get some insights on this, consider the regression plots in Figure 3.6, comparing, for a network with $H_1 = H_2 = 25$ hidden neurons, some of the components of the network output with the correspondent targets on Ξ_{te} . While the first and seventh components (*top*) are interpolated almost exactly, problems start arising for the eighth (*bottom left*) and mainly for the ninth (*bottom right*) entries. This suggests that the Levenberg-Marquardt algorithm is sufficiently *smart* to detect that the average magnitude of the components of the target vectors decreases with the ordinality of the components themselves¹, and so that particular effort should be put in a proper regression of the first components. Indeed, although the prediction for the ninth entry of any test vector $\mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu})$ is completely off in most of the cases, the average magnitude of these components is small - around 10^{-4} . As a result, the global solution provided by the network turns out to be really close to the correspondent FE discretization, as shown in Figure 3.5 (*right*).

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

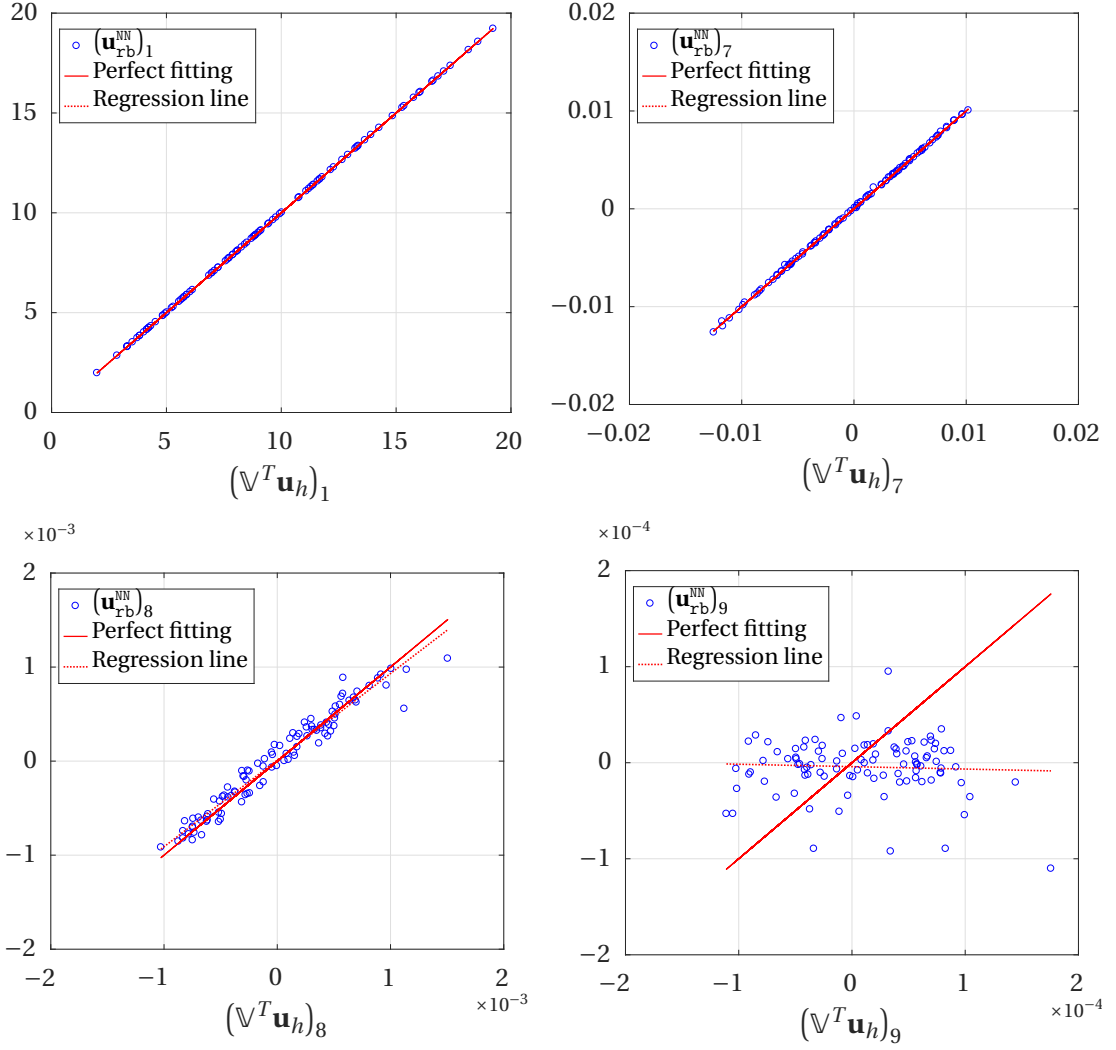


Figure 3.6. From left to right, top to bottom: regression plots for the first, seventh, eighth and nine scalar components of the outputs provided on Ξ_{te} by a network with $H_1 = H_2 = 25$ neurons per hidden layer, trained to approximate the map (2.91) for problem (3.5).

3.1.3 Nonlinear test case, three parameters

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

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

with $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3) \in \mathcal{P} = [1, 3] \times [1, 3] \times [-0.5, 0.5]$ and $s(\cdot; \boldsymbol{\mu})$ such that $u_{ex}(x; \boldsymbol{\mu}) = \mu_2 (2 + \sin(\mu_1 x)) \exp(\mu_3 x)$ is the exact solution to the problem for any $\boldsymbol{\mu} \in \mathcal{P}$. Figure 3.7 represents the first 10 POD modes, yielded via singular value decomposition of an ensemble of $N = 100$ (linear) FE snapshots computed over a uniform grid consisting of 100 nodes. Their inclusion within the standard POD-Galerkin RB framework enables an average relative error of about

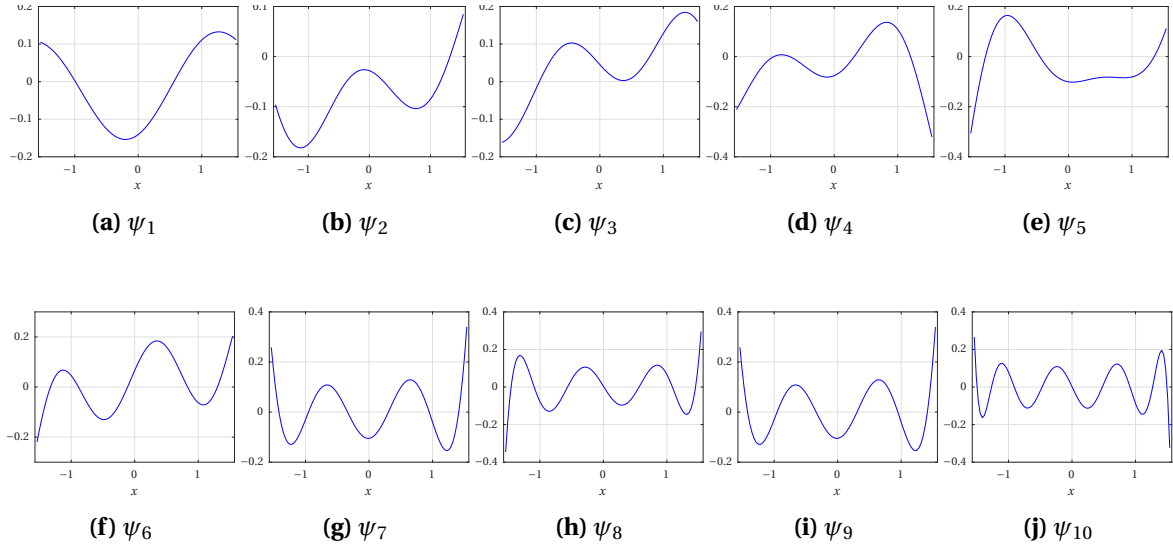


Figure 3.7. The first 10 POD basis functions for the (nonlinear) Poisson problem (3.6).

10^{-5} on a test set Ξ_{te} of $N_{te} = 100$ randomly chosen samples. This is witnessed by Figure 3.8, offering a convergence analysis with respect to the number of basis functions retained in the model (*left*). The results for the POD-NN method, employing three-layers neural networks with a varying number of neurons and learning patterns, are provided as well, together with a comparison with the FE solutions for some parameter values (*right*). Note that, with only a few modes are considered (say $L \leq 6$), the proposed reduced basis scheme leads to a significant reduction of the error with respect to the POD-RB technique. Moreover, as we expand the training set, we get more accurate predictions for a larger number of POD coefficients, provided that we allow the size of the network to increase.

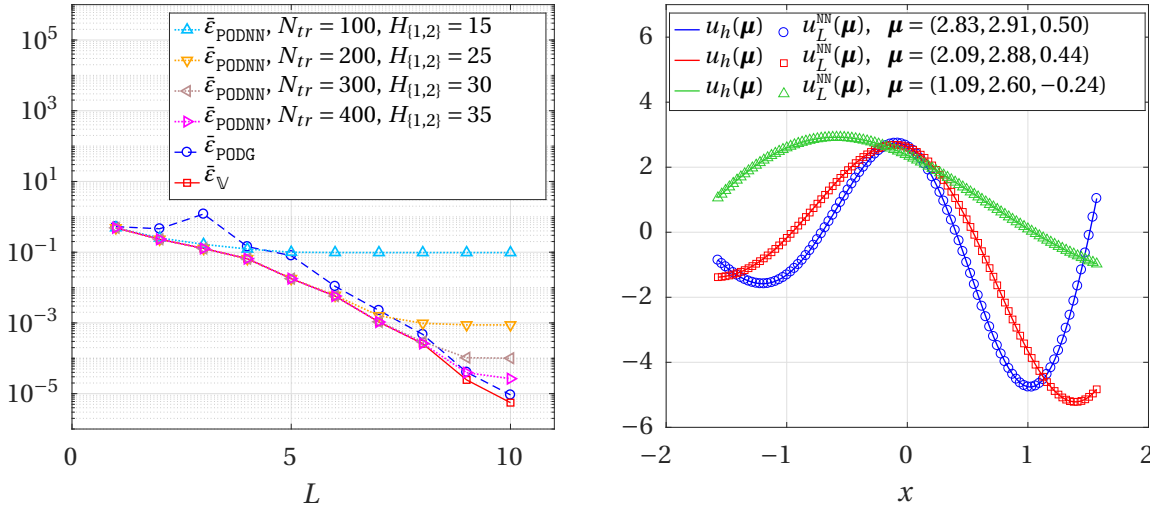


Figure 3.8. Convergence analysis for the POD-G and POD-NN methods applied to the problem (3.6) (*left*) and comparison between the FE and the POD-NN solutions for three parameter values (*right*). These latter results have been obtained via a neural network with $H_1 = H_2 = 35$ units per hidden layer,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

Concerning the results shown in the left plot of Figure 3.8, we should point out an important remark, which shall be valid throughout the chapter. The results shown in Figure 3.8 refer to neural networks trained via exposition to the set of coefficients $\{(u_h(\boldsymbol{\mu}), \psi_i)_h\}_{i=1}^{\bar{L}} \equiv \{(\mathbf{u}_h(\boldsymbol{\mu}), \boldsymbol{\psi}_i)_{\mathbb{R}^M}\}_{i=1}^{\bar{L}}$ per learning sample $\boldsymbol{\mu} \in \Xi_{tr}$, with $\bar{L} = 10$. Therefore, each network is equipped with $\bar{L} = 10$ output neurons - one for each coefficient. Once the training has been successfully completed, given a new input $\boldsymbol{\mu} \in \mathcal{P}$, the correspondent POD-NN RB solution of order \bar{L} is obtained as:

$$\mathbf{u}_{\bar{L}}^{\text{NN}}(\boldsymbol{\mu}) = \sum_{i=1}^{\bar{L}} (\mathbf{u}_{\bar{L}}^{\text{NN}})_i \boldsymbol{\psi}_i, \quad (3.7)$$

where $(\mathbf{u}_{\bar{L}}^{\text{NN}})_i$ denotes the network response observed at the i -th output neuron, $i = 1, \dots, \bar{L}$. Although $\mathbf{u}_{\bar{L}}^{\text{NN}}(\boldsymbol{\mu})$ represents the best prediction for $\mathbf{u}_h(\boldsymbol{\mu})$ provided by the network, we can as well construct lower-ordered approximations $\mathbf{u}_L^{\text{NN}}(\boldsymbol{\mu})$, with $L < \bar{L}$ - simply retain the first L terms in the summation (3.7). The claim is that such an L -order RB solution is basically as accurate as the one we could obtain by exposing the network to exactly L POD coefficients per training sample. In other terms, employing a larger amount of modes during the training than in the subsequent online queries, so somehow offering the network a broader overview on the problem, do not enable a sensible improvement in the performance. A confirmation of this is given by Figure 3.9, reporting the average relative POD-NN error $\bar{\epsilon}_{\text{NN}}(L)$, for $1 \leq L \leq 10$ yielded by networks equipped with either 20 or 35 units per hidden layer, and exposed, within the learning step, to $N_{tr} = 300$ samples and either L or \bar{L} associated POD coefficients.

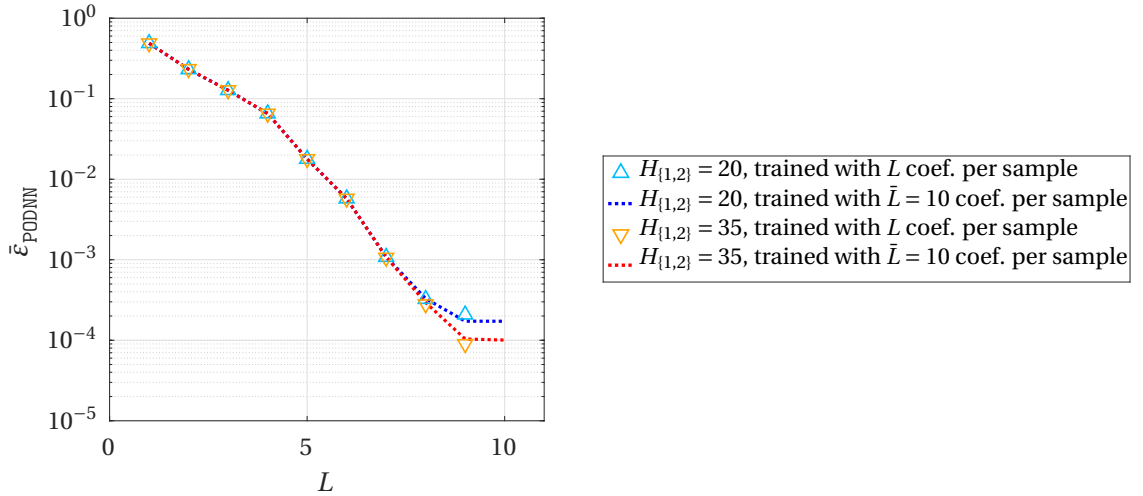


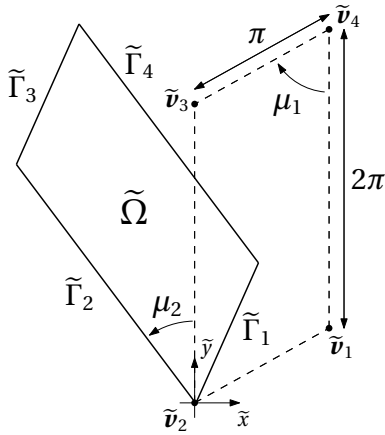
Figure 3.9. Sensitivity analysis for the POD-NN method applied to the problem (3.6) with respect to the number of POD coefficients (per sample) used during the training; to prevent overfitting, $N_{tr} = 300$ learning patterns have been used.

3.2 Two-dimensional Poisson equation

As disclosed in the introductory part, the two-dimensional differential problems we deal with in this chapter uniquely involve a geometrical parametrization, that is, $\boldsymbol{\mu} \equiv \boldsymbol{\mu}_g$. In particular, in this section we address the numerical discretization via RB methods of three

BVPs concerning the Poisson equation, either linear or nonlinear. Each problem will be stated over one of the two domains $\tilde{\Omega}(\boldsymbol{\mu})$ depicted in Figures 3.10 and 3.11. The former consists of a parallelogram, possibly rotated around the origin, and it is characterized by two parameters: one of the internal angles, $\mu_1 \in [\pi/4, 3\pi/4]$, and the rotation angle itself, $\mu_2 \in [0, \pi/2]$. Whereas, the latter represents a stenosis geometry, parametrized in the depths μ_1 and μ_2 of the bottom and top restrictions (or inflations), respectively, and the length μ_3 of the vessel; the parameter domain is given by $\mathcal{P} = [-0.5, 0.5] \times [-0.5, 0.5] \times [1, 5]$. For both geometries, the boundary parametrizations are reported alongside their visualization.

As discussed in Section 2.2, in order to generate snapshots which can then be combined, the problems have to be re-formulated and solved on a reference, i.e., parameter-independent, configuration Ω , discretized through a triangular mesh Ω_h via M internal vertices. Here, Ω is given by the unit square shown in Figure 2.2, while the volumetric transformation $\Phi : \Omega \times \mathcal{P} \rightarrow \tilde{\Omega}$ used to transpose the PDE from the physical to the fixed domain is provided by the boundary displacement-dependent transfinite map (2.41), built from the parametrizations of the edges of $\tilde{\Omega}(\boldsymbol{\mu})$ (see Section 2.2.3 for further details).



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

$$\tilde{\mathbf{v}}_1 = \tilde{\mathbf{v}}_1(\mu_1) = [\pi \sin(\mu_1), \pi \cos(\mu_1)]^T$$

$$\tilde{\mathbf{v}}_2 = \tilde{\mathbf{v}}_2(\mu_1) \equiv [0, 0]^T$$

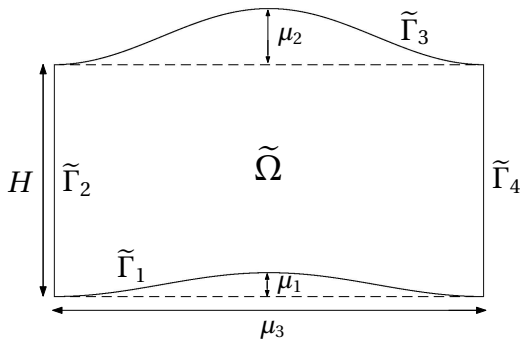
$$\tilde{\mathbf{v}}_3 = \tilde{\mathbf{v}}_3(\mu_1) \equiv [0, 2\pi]^T$$

$$\tilde{\mathbf{v}}_4 = \tilde{\mathbf{v}}_4(\mu_1) = [\pi \sin(\mu_1), \pi(2 + \cos(\mu_1))]^T$$

$$\tilde{\Psi}_i(t; \boldsymbol{\mu}) = \mathbb{Q}(\mu_2) [t \tilde{\mathbf{v}}_i + (1-t) \tilde{\mathbf{v}}_{i+1}]$$

for $1 \leq i \leq 4$, with $\tilde{\mathbf{v}}_5 \equiv \tilde{\mathbf{v}}_1$

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



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

$$\tilde{\Psi}_2(t; \boldsymbol{\mu}) \equiv [0, Ht]^T$$

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

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

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

3.2.1 Linear test case

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

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

Once the problem (3.8) has been re-stated over the reference domain Ω by means of the map (2.42) and the change of variables formulae reported in Section 2.2.1, the discretization via finite elements yields the linear system

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

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

$$\begin{aligned} \mathbb{A}(\boldsymbol{\mu})_{i,j} &= \sum_{K \in \Omega_h} a_K(\phi_j, \phi_i) = \int_K \mathbb{J}_{\Phi}^{-T}(\boldsymbol{\mu}) \nabla \phi_j \cdot \mathbb{J}_{\Phi}^{-T}(\boldsymbol{\mu}) \nabla \phi_i |\mathbb{J}_{\Phi}(\boldsymbol{\mu})| dK \quad \text{for } i, j = 1, \dots, M, \\ \mathbf{s}(\boldsymbol{\mu})_i &= \sum_{K \in \Omega_h} \int_K (\tilde{s} \circ \Phi) \phi_i |\mathbb{J}_{\Phi}(\boldsymbol{\mu})| dK - \sum_{K \in \Omega_h} a_K(l_h(\boldsymbol{\mu}), \phi_i) \quad \text{for } i = 1, \dots, M. \end{aligned}$$

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

$$\mathbb{A}_L(\boldsymbol{\mu}) \mathbf{u}_{rb}(\boldsymbol{\mu}) = \mathbf{s}_L(\boldsymbol{\mu}) \quad (3.10)$$

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

$$\mathbb{A}_L(\boldsymbol{\mu}) = \mathbb{V}^T \mathbb{A}(\boldsymbol{\mu}) \mathbb{V} \in \mathbb{R}^{L \times L} \quad \text{and} \quad \mathbf{s}_L(\boldsymbol{\mu}) = \mathbb{V}^T \mathbf{s}(\boldsymbol{\mu}) \in \mathbb{R}^L. \quad (3.11)$$

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

automatic routine 2.5, performed to find an optimal combination of the number of training samples and the amount of hidden neurons to use, takes around 2 hours, as opposed to the few minutes required to build $N = 100$ FE snapshots (through the resolution of the system (3.9)) and compute their singular value decomposition. However, coupling POD with a neural network with $H_1 = H_2 = 35$ computing units per hidden layer and whose weights are adjusted relying upon $N_{tr} = 200$ training patterns $\{\boldsymbol{\mu}^{(i)}, \mathbb{V}^T \mathbf{u}_h(\boldsymbol{\mu}^{(i)})\}_{i=1}^{N_{tr}}$, with $\mathbb{V} \in \mathbb{R}^{M \times 15}$, yields (actually slightly) more accurate results than the POD-G method does, as can be inferred from Figure 3.13 (*right*). In particular, the POD-NN error $\bar{\epsilon}_{\text{PODNN}}(L)$ matches the projection error $\bar{\epsilon}_{\mathbb{V}}(L)$ for any $L \leq 15$; a convergence analysis for $L = 15$ is given in the plot on the left.

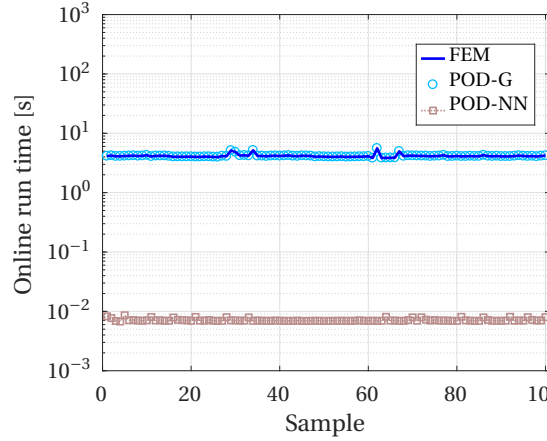


Figure 3.12. Online run times for the FE, POD-G and POD-NN methods applied to the problem (3.8), for $N_{te} = 100$ randomly generated parameter values.

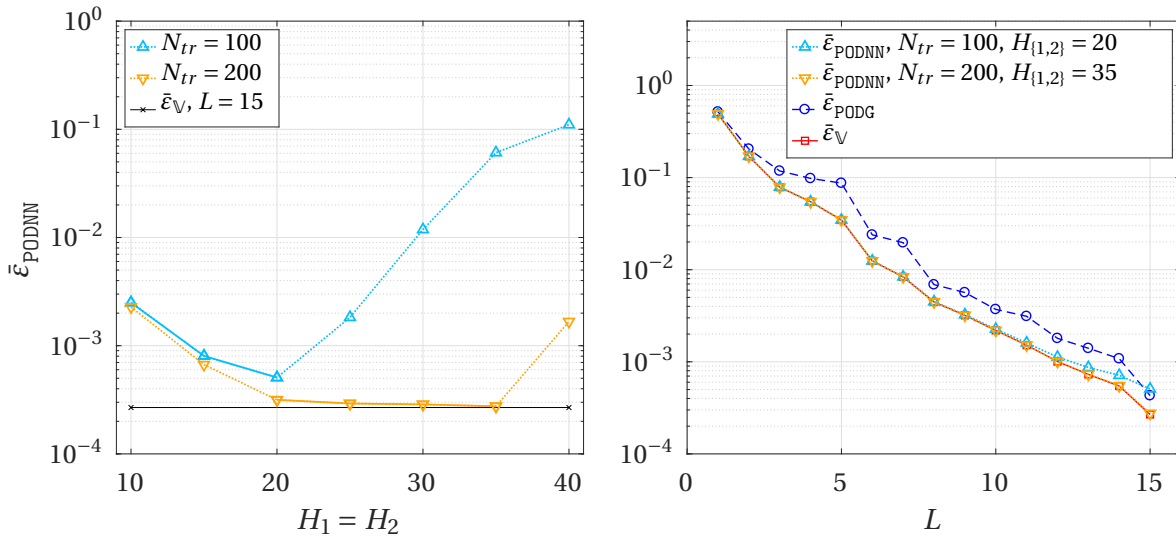


Figure 3.13. On the right, error analysis for the POD-G (blue dashed line) and the POD-NN (dotted lines) methods applied to the problem (3.8). For the latter, a convergence analysis with respect to the number of training samples and hidden neurons used is provided on the left. The solid tracts denote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

3.2.2 Nonlinear test case, two parameters

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

$$\begin{cases} -\tilde{\nabla} \cdot ((1 + \tilde{u}(\boldsymbol{\mu})^2) \tilde{\nabla} \tilde{u}(\boldsymbol{\mu})) = \tilde{s}(\tilde{x}, \tilde{y}) & \text{in } \tilde{\Omega}(\boldsymbol{\mu}), \\ \tilde{u}(\boldsymbol{\mu}) = \tilde{\sigma}_x \sin(\tilde{\sigma}_x) \cos(\tilde{\sigma}_y) & \text{on } \partial\tilde{\Omega}(\boldsymbol{\mu}). \end{cases} \quad (3.12)$$

Here, the source term $\tilde{s} = \tilde{s}(\tilde{x}, \tilde{y})$ has been chosen so that for any $\boldsymbol{\mu} \in \mathcal{P} = [\pi/4, 3\pi/4] \times [0, \pi/2]$ the exact solution to the problem is given by $\tilde{u}_{ex}(\tilde{x}, \tilde{y}) = \tilde{y} \sin(\tilde{x}) \cos(\tilde{y})$. We have already referred to this problem in Section 2.6, as it lays bare the major weak points of the standard POD-Galerkin framework. In fact, as witnessed by Figure 2.4, the Newton's method applied to the (nonlinear) reduced system (2.67) might fail to converge here when the number of modes included in the model is sufficiently large, say greater than 25. In this regard, Figure 3.14 compares the truth (*left*), POD-G (*center*) and POD-NN (*right*) solutions to the problem (3.12) for $\boldsymbol{\mu} = (0.835, 0.034)$. The FE discretization has been built over a mesh consisting of $M = 1921$ nodes, whereas the RB solutions have been constructed combining $L = 33$ POD basis functions. While the POD-NN procedure, employing $H_1 = H_2 = 35$ hidden neurons and $N_{tr} = 300$ learning patterns, provides a (qualitatively) good approximation, the discrepancy between the FE and POD-G solutions is evident. Indeed, the selected parameter value coincides with the highest peak of the dotted curve in the left plot of Figure 3.15, which reports the POD-G error (3.1) and the POD-NN error (3.2) for $N_{te} = 50$ randomly picked samples not involved either in the generation of the reduced basis, nor in the networks training. Generally, resorting to neural networks not only yields more precise responses, but also drastically reduces the time required to tackle any online query, as shown by the graph on the right in Figure 3.15.

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

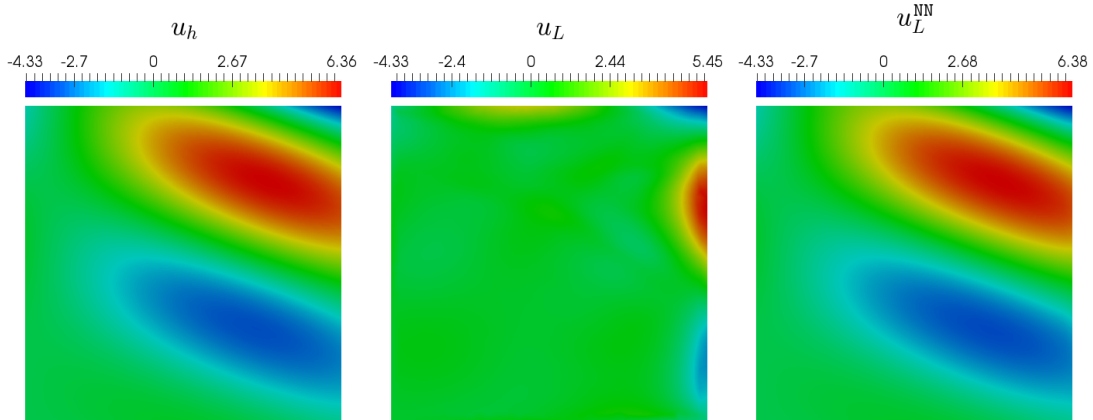


Figure 3.14. Finite element (*left*), POD-G (*center*) and POD-NN (*right*) solution to the semilinear Poisson problem (3.12) re-stated over the reference domain, with $\boldsymbol{\mu} = (0.835, 0.034)$.

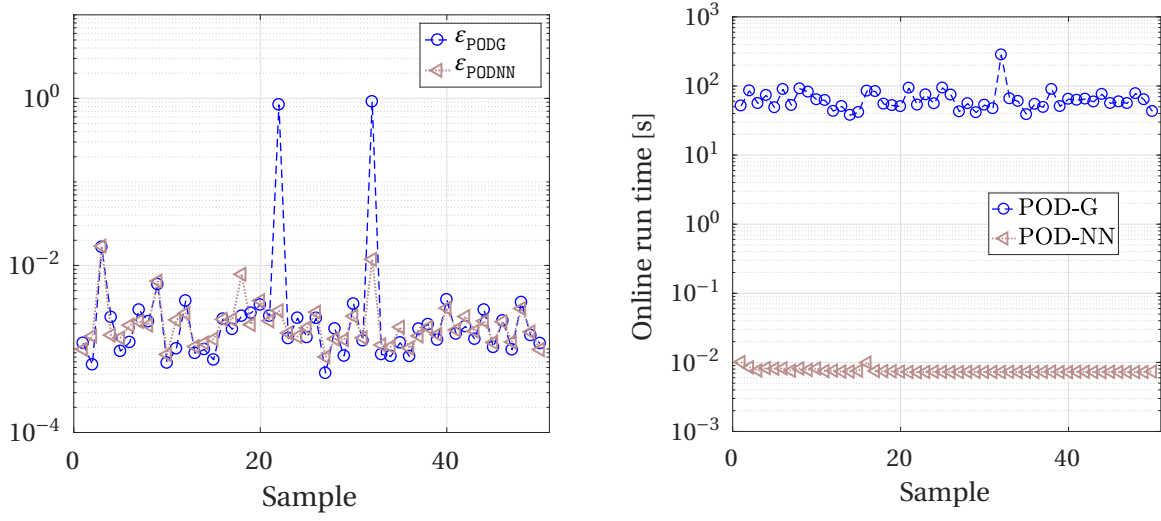


Figure 3.15. Online relative errors (*left*) and run times (*right*) for the POD-G and the POD-NN methods applied to problem (3.12) for $N_{tr} = 50$ randomly picked parameter values.

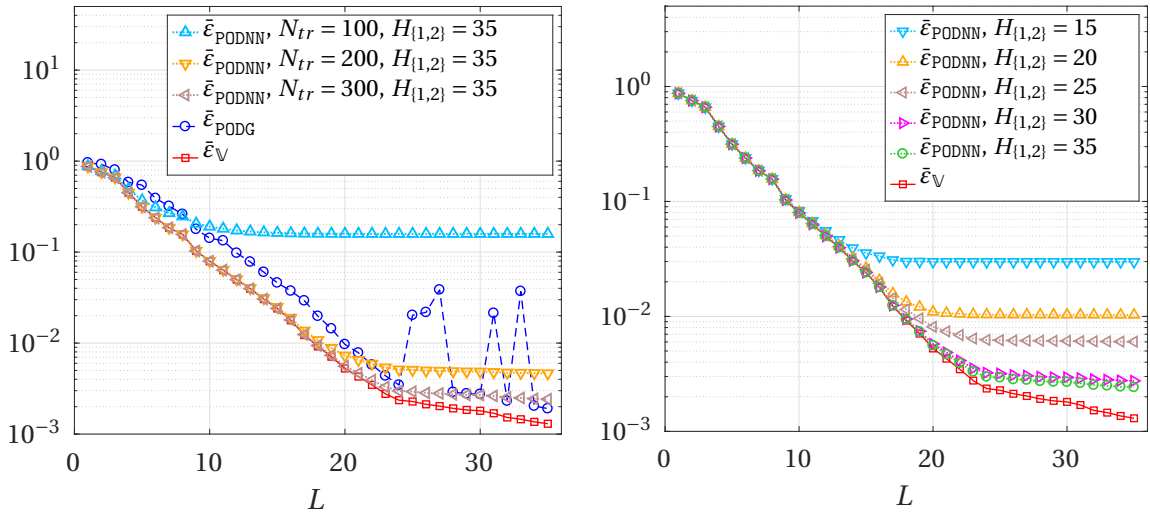


Figure 3.16. Convergence analysis for the POD-NN RB method applied to the BVP (3.12). The results have been obtained via three-layers (i.e., two hidden layers plus the output layer) neural networks with $H_1 = H_2 \in \{15, 20, 25, 30, 35\}$ neurons in each hidden layer, and trained with $N_{tr} \in \{100, 200, 300\}$ learning patterns. Each pattern consists of an input vector $\boldsymbol{\mu} \in \mathcal{P}$ and the first $L = 35$ POD coefficients $(\mathbf{u}_h(\boldsymbol{\mu}), \boldsymbol{\psi}_i)_{\mathbb{R}^M}$, $i = 1, \dots, L$, as teaching inputs.

3.2.3 Nonlinear test case, three parameters

For the last nonlinear test case concerning the Poisson equation, we consider again the stenosis geometry of Figure ??, with $\mu_1, \mu_2 \in [-0.5, 0.5]$ and $\mu_3 \in [1, 5]$. The differential

problem we deal with in this section features an exponential nonlinearity, and reads:

$$\begin{cases} -\tilde{\nabla} \cdot (\exp(\tilde{u}(\boldsymbol{\mu})) \tilde{\nabla} \tilde{u}(\boldsymbol{\mu})) = \tilde{s}(\tilde{x}, \tilde{y}) & \text{in } \tilde{\Omega}(\boldsymbol{\mu}), \\ \tilde{u}(\boldsymbol{\mu}) = \tilde{\sigma}_x \sin(\pi \tilde{\sigma}_x) \cos(\pi \tilde{\sigma}_y) & \text{on } \partial \tilde{\Omega}(\boldsymbol{\mu}). \end{cases} \quad (3.13)$$

with the source term $\tilde{s} = \tilde{s}(\tilde{x}, \tilde{y})$ properly chosen so that the exact solution to the problem is given by $\tilde{u}_{ex}(\tilde{x}, \tilde{y}) = \tilde{y} \sin(\pi \tilde{x}) \cos(\pi \tilde{y})$ for all $\boldsymbol{\mu} \in \mathcal{P}$.

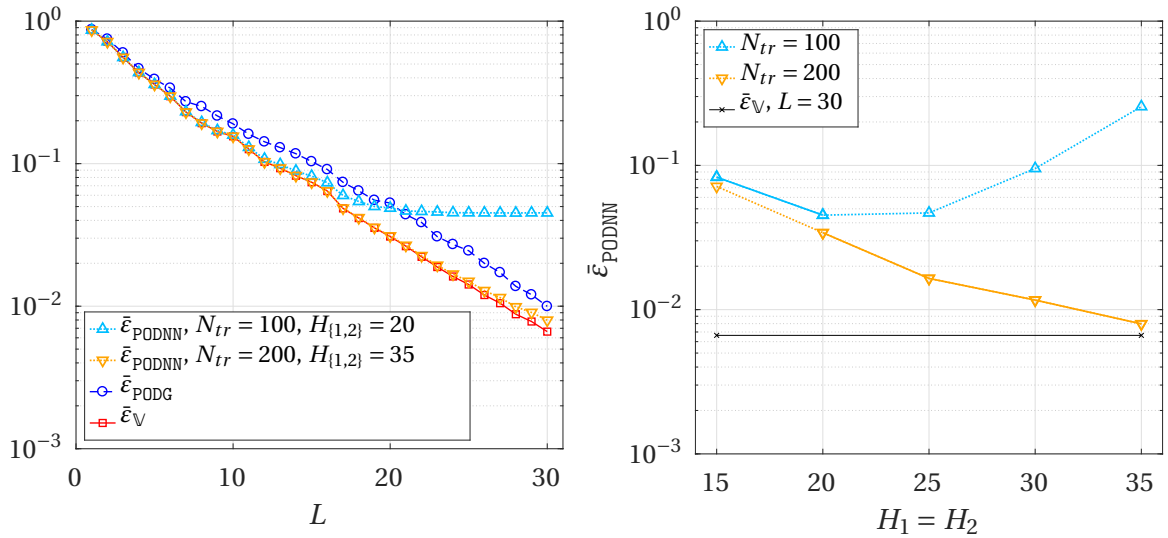


Figure 3.17. On the left, error analysis for the POD-G (blue dashed line) and the POD-NN (dotted lines) methods (*left*) applied to the problem (3.13). For the latter, a convergence analysis with respect to the number of training samples and hidden neurons used is provided on the right. The solid tracts denote the steps actually pursued by the automatic training routine 2.5. All the results refer to neural networks exposed to $L = 30$ POD coefficients per sample during the learning step.

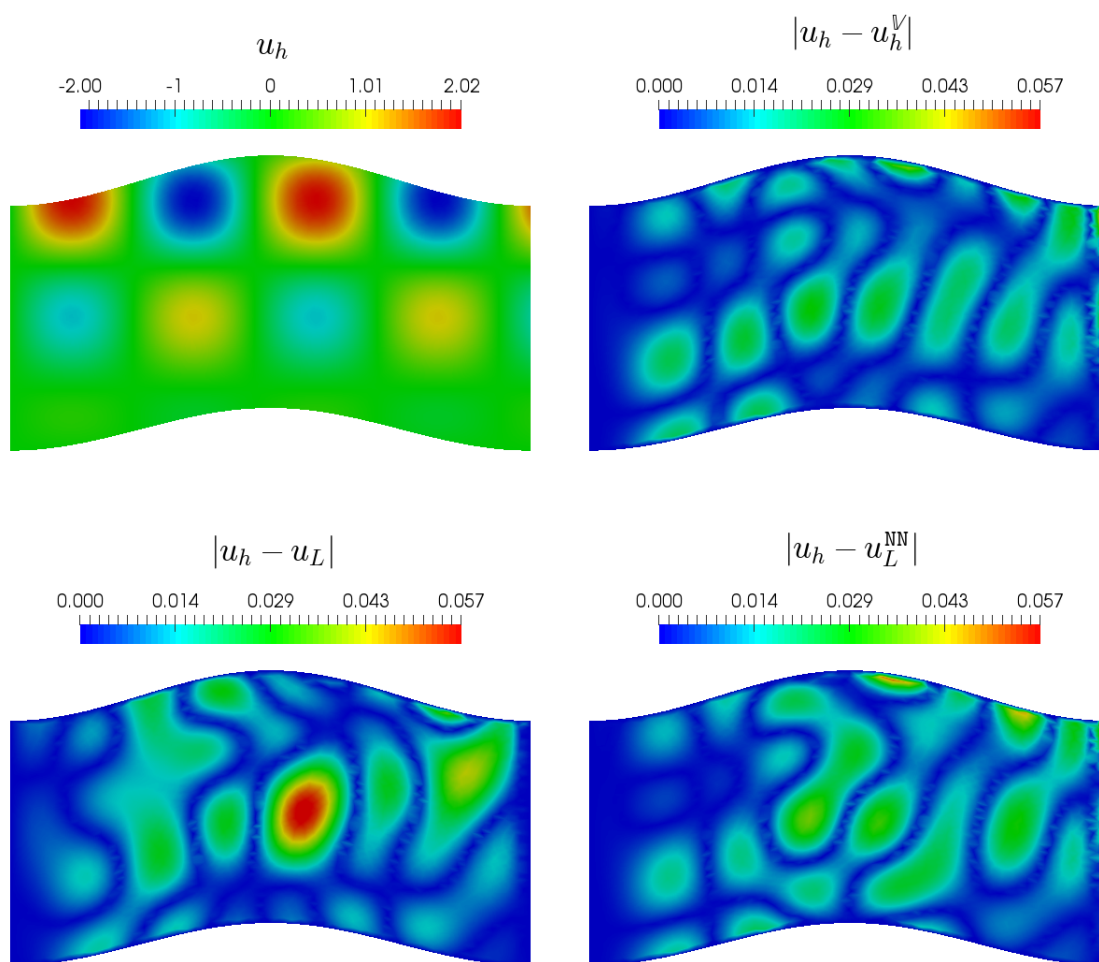


Figure 3.18

Bibliography

- [1] Amsallem, D. (2010). *Interpolation on manifolds of CFD-based fluid and finite element-based structural reduced-order models for on-line aeroelastic predictions*. Doctoral dissertation, Department of Aeronautics and Astronautics, Stanford University.
- [2] Ballarin, F., Manzoni, A., Quarteroni, A., Rozza, G. (2014). *Supremizer stabilization of POD-Galerkin approximation of parametrized Navier-Stokes equations*. MATHICSE Technical Report, École Polytechnique Fédérale de Lausanne.
- [3] Barrault, M., Maday, Y., Nguyen, N. C., Patera, A. T. (2004). *An 'empirical interpolation' method: Application to efficient reduced-basis discretization of partial differential equations*. Comptes Rendus Mathématique, 339(9):667-672.
- [4] 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.
- [5] 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 Engineering, 196:337-355.
- [6] Caloz, G., Rappaz, J. (1997). *Numerical analysis and bifurcation problems*. Handbook of numerical analysis, 5(2):487-637.
- [7] 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.
- [8] Cybenko, G. (1988). *Continuous valued neural networks with two hidden layers are sufficient*. Technical Report, Department of Computer Science, Tufts University.
- [9] Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems, 2(4):303-314.
- [10] 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.
- [11] 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.

- [12] Fahlman, S. E. (1988). *An empirical study of learning speed in back-propagation networks*. Technical Report CMU-CS-88-162, CMU.
- [13] Gill, P. E., Murray, W., Wright, M. H. (1981). *Practical optimization*. Academic Press.
- [14] Hagan, M. T., Menhaj, M. B. (1994). *Training feedforward networks with the Marquardt algorithm*. IEEE Transactions on Neural Networks, 5(6):989-993.
- [15] 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>.
- [16] Hassdonk, B. (2013). *Model reduction for parametric and nonlinear problems via reduced basis and kernel methods*. CEES Computational Geoscience Seminar, Stanford University.
- [17] Haykin, S. (2004). *Neural Networks: A comprehensive foundation*. Upper Saddle River, NJ: Prentice Hall.
- [18] Hebb, D. O. (1949). *The organization of behaviour: A neuropsychological theory*. New York, NY: John Wiley & Sons.
- [19] 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.
- [20] Hassibi, B., Stork, D. G. (1993). *Second order derivatives for network pruning: Optimal Brain Surgeon*. Advances in neural information processing systems, 164-171.
- [21] Hesthaven, J. S., Starn, B., Rozza, G. (2016). *Certified reduced basis methods for parametrized partial differential equations*. New York, NY: Springer.
- [22] Hopfield, J. J. (1982). *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the National Academy Science, 79:2554-2558.
- [23] Imam, R. L. (2008). *Latin hypercube sampling*. Encyclopedia of Quantitative Risk Analysis and Assessment.
- [24] 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.
- [25] Kaelbling, L. P., Littman, M. L., Moore, A. W. (1996). *Reinforcement Learning: A Survey*. Journal of Artificial Intelligence Research, 4:237-285.
- [26] Kohavi, R. (1995). *A study of cross-validation and bootstrap for accuracy estimation and model selection*. Proceedings of the 40th International Joint Conference on Artificial Intelligence, 2(12):1137-1143.
- [27] Kohonen, T. (1998). *The self-organizing map*. Neurocomputing, 21(1-3):1-6.

- [28] Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. Retrieved from http://www.dkriesel.com/en/science/neural_networks.
- [29] 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.
- [30] 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.
- [31] 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>.
- [32] Mitchell, W., McClain, M. A. (2010). *A collection of 2D elliptic problems for testing adaptive algorithms*. NISTIR 7668.
- [33] Manzoni, A., Negri, F. (2016). *Automatic reduction of PDEs defined on domains with variable shape*. MATHICSE technical report, École Polytechnique Fédérale de Lausanne.
- [34] Moller, M. D. (1993). *A scaled conjugate gradient algorithm for fast supervised learning*. Neural Networks, 6:525-533.
- [35] McClelland, J. L., Rumelhart, D. E. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, UK: MIT Press.
- [36] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.
- [37] 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.
- [38] Persson, P. O. (2002). *Implementation of finite-element based Navier-Stokes solver*. Massachussets Institue of Technology.
- [39] 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.
- [40] Quarteroni, A. (2010). *Numerical models for differential problems* (Vol. 2). New York, NY: Springer Science & Business Media, 2010.
- [41] Quarteroni, A., Manzoni, A., Negri, F. (2015). *Reduced basis methods for partial differential equations: An introduction* (Vol. 92). New York, NY: Springer, 2015.
- [42] Rannacher, R. (1999). *Finite element methods for the incompressible Navier-Stokes equations*. Lecture notes, Institute of Applied Mathematics, University of Heidelberg.
- [43] Riedmiller, M., Braun, H. (1993). *A direct adaptive method for faster backpropagation learning: The rprop algorithm*. Neural Networks, IEEE International Conference on, 596-591.

-
- [44] Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 65:386-408.
 - [45] Rudin, W. (1964). *Principles of mathematical analysis* (Vol. 3). New York, NY: McGraw-Hill.
 - [46] Stergiou, C., Siganos, D. (2013). *Neural Networks*. Retrieved from https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introductiontoneuralnetworks.
 - [47] Volkwein, S. *Model reduction using proper orthogonal decomposition*. Lecture notes.
 - [48] Widrow, B., Hoff, M. E. (1960). *Adaptive switching circuits*. Proceedings WESCON, 96-104.