ARISTOTLE UNIVERSITY OF THESSALONIKI

Reduced Basis Methods for PDEs and Digital Twins

by
Sapranidis Ioannis

A thesis submitted in partial fulfillment for the
Master's program
Theoretical Computer Science & Systems and Control Theory

Faculty of Natural Sciences
Department of Mathematics

Supervising Professor: Dr. Karatzas Efthymios

January 2025

Declaration on Intellectual Property Rights

I, Ioannis Sapranidis, declare that this thesis with title "Reduced Basis Methods for PDEs and Digital Twins", and the work presented in it, are mine and mine only. I confirm that:

- This work took place completely or mostly during my candidacy for the title of master's degree in this specific university.
- Wherever any part of this thesis has been previously filed for the acquisition of a bachelor's degree or any other diploma in this or any other university, it is clearly expressed.
- Wherever I have consulted published work of a third party, it is attributed correctly.
- Wherever I have quoted part of published work of a third party, the source is always stated. With the exception of these citations, this thesis is completely my personal work product.
- I have quoted all sources of help.
- Wherever this thesis is based on work produced by a cooperation between me and a third party, it is clearly stated which parts are produced by others and my percentage of contribution to them.

Mathematics exists solely for the honor of the human mind.

Carl Gustav Jacob Jacobi

Reduced Basis Methods for PDEs and Digital Twins

ARISTOTLE UNIVERSITY OF THESSALONIKI
Faculty of Natural Sciences
Department of Mathematics

by
Sapranidis Ioannis

Abstract

This thesis presents a study on numerical ways for the efficient visualization of fluid problems through the prism of Partial Differential Equations. This study focuses mainly on the review and comparison between the Finite Element Methods, Reduced Basis Methods, and also on the construction and function of Digital Twins such as Physics Informed Neural Networks (PINNs), Deep Learning Operator Networks (DeepONet), etc. as recent and very useful applications of technology. The thesis starts with an introduction to the generalized category of Fluid problems. In the first part, we investigate the Finite Elements Method as a classic method for the approximation to the solutions of the partial differential equations. We look into the creation of the mesh, the discretization of the function spaces, and the algorithm that is implemented in order to approximate the solutions. In the second part, we introduce the Reduced Basis Methods which provide us with a more efficient way for the reduction of the computational complexity by constructing models of a lower dimension. The study here investigates the construction and implementation of these models, which are presented in the second chapter. In the third part, we explore the structure, the function, and the construction of Digital Twins. The latter are known to the world of science as a newfound technology that is mainly used in solving complex problems and optimizing procedures. The thesis analyzes the methods used by the Digital Twins as well as the procedures followed to accomplish the desired result in the best time possible. Lastly, we state the importance of Digital Twins in modern daily life, and a conversation is held around the targets that can be actualized through them in the following years.

Dedicated
to my parents, Sokratis and Doukeni,
and to my lovely girlfriend, Anna.

Contents

Chapter 1

Introduction

In recent years, advancements in computational methods have transformed how we approach complex scientific and engineering problems, particularly those governed by Partial Differential Equations (PDEs). This thesis explores advanced methods that aim at achieving a gradually increased efficiency in the way that PDEs are solved, considering factors such as data availability, computational resources (time, complexity, etc.), and visualized for a better understanding of the geometry of the domain and the solutions in question.

Finite Element Methods (FEM) [4] were developed in the 1950s and 1960s to address difficult structural engineering issues in aerospace and civil engineering by dividing continuous systems into smaller, more manageable components. This technology enabled engineers to estimate solutions for systems that were too complicated for analytical approaches, revolutionizing computational mechanics. Today, FEM is commonly utilized in engineering, physics, and biological sciences. Applications include structural analysis of buildings and bridges, fluid dynamics, heat transfer, and electromagnetics. FEM is also used in digital twin technologies to do real-time simulations and optimizations. FEM's versatility and precision make it an essential tool for both studying and practical problem-solving in complex systems. This way, in the first chapter, we describe the basis of FEM: piecewise polynomial functions that approximate complex functions through meshing. We introduce function spaces within which our solutions lie, along with the Galerkin method and the Lax-Milgram Lemma, both important components to establish the existence, uniqueness, and optimization of the solution within a finite subspace, the geometry in question.

In the second chapter, Reduced Order Methods (ROMs) [1] are introduced, which were developed to address the computational challenges associated with solving parametrized PDEs in scenarios requiring repeated simulations, like optimization and real-time control. By constructing a low-

dimensional approximation of the solution space, ROMs dramatically reduce computational costs while retaining essential solution features, making them particularly valuable for complex, high-dimensional problems that require rapid response times. They are applied in fluid dynamics, structural analysis, and electromagnetics, among other areas, where reducing the computational load is essential for iterative and high-speed processes. We state different, efficient, each in its own way and field, methodologies that take as input the original spaces that describe each phenomenon and, with controlled tolerance, produce low-dimension, corresponding spaces that capture better and faster the high-fidelity underlying problem.

Both FEM and ROMs meet extensive use in Digital Twins [7], which are virtual replicas of physical systems or processes. They integrate real-time data from sensors, enabling simulations and predictive analysis. The primary innovation lies in their ability to continuously update, adapt, and optimize based on real-world performance. Applications span across manufacturing, healthcare (e.g., patient monitoring), urban planning, automotive (predictive maintenance), and energy systems. We begin by stating the basic structure of a Digital Twin, giving explicit details for each section while using previously stated information from FEM and ROMs to show the profound role each of them plays in it. Since these two methods may prove somewhat inefficient in cases of data shortage, high complexity, and computational demand, modern-day improvement of Machine Learning arises to give a relative solution to these problems by providing additional data, lowering the complexity, or simplifying the procedures in question. This way, structures are formed that combine both procedures that solve PDEs and ones that optimize the results using scientific methods of Machine Learning, which divide into three main categories: Physics-Induced Neural Networks, Deep Energy Methods, and Neural Operators. Each of these offers several advantages in different cases of problems and combines diverse proportions of physics, such as the mathematical interpretation of a phenomenon, and available data, based on measurements made in real or semi-real time.

Chapter 2

Finite Element Method

In this chapter we set the knowledge basis, that is, partly already known from calculus and algebra, fitted in the case of the Finite Element Method (FEM) [4]. This way, we introduce the piecewise polynomial functions that are used to approximate more complicated and general functions and are easier to implement into computer software. We will mainly focus on the extension of this concept in two dimensions and will give a definition for the Finite Elements based on how we construct these piecewise polynomial functions upon the underlying domain. Before we do that, we have to divide the domain into elements, such as triangles, and present a basic methodology for the triangulation and the mesh generation. Finally, we will talk about the main implementation of the FEM, and provide sources for some novel examples of implementing the aforementioned procedure.

2.1 Piecewise Polynomial Functions

In this section, we set the grounds for the Piecewise Polynomial Functions but in order to define them, we need first to establish a way to "divide" our two-dimensional space into pieces, or elements, a procedure called meshing, and to properly define the space of the aforementioned functions as well as their structure and properties [4].

2.1.1 Meshes - Triangulation

Let $\Omega \subset \mathbb{R}^2$ be a domain of two dimensions with its boundary $\partial\Omega$ to be smooth or polygonal.

Definition 2.1.1. *A triangulation, or mesh, $\mathcal{K}$ of Ω is a set of triangles K, denoted as $\{K\}$, such that $\Omega = \bigcup_{K\in\mathcal{K}} K$, so that the intersection of two*

triangles is either an edge, a corner, or an empty space. The corners of the triangles are called nodes.

We will denote the set of triangle edges E by $\mathcal{E} = E$. We will separate the set into two subsets, one that contains the edges that lie within the domain Ω, or interior edges $\mathcal{E}_I$, and one that contains the edges that lie on the boundary $\partial\Omega$, or boundary edges $\mathcal{E}_B$.

At this point, we can introduce a way to measure the size of triangle K as local mesh size h_k, which is the length of the longest edge in K. We also measure the quality of K by using the diameter of the inscribed circle or d_K so that we can define finally the chunkiness parameter c_K, by

$$c_K = \frac{h_K}{d_K}. \tag{2.1}$$

We can characterize a triangulation $\mathcal{K}$ as a shape regular if there is a real constant $c_0 > 0$ such that

$$c_K \geq c_0, \quad \forall K \in \mathcal{K}. \tag{2.2}$$

Supposing this condition applies, the shape of the triangles isn't too extreme angle-wise, meaning that all the angles are nor very wide or very narrow.

We define global mesh size as $h = \max_{K\in\mathcal{K}} h_K$, and we will mainly refer to quasi-uniform meshes, which means that the mesh size h_K is roughly the same for all our meshes. We use a constant $\rho > 0$ s.t. $\rho < \frac{h_K}{h_{K'}} < \rho^{-1}$, for any two different elements K and K'.

Assuming that the meshes we work with from now on are shape regular, we can define piecewise polynomial spaces and their approximation properties on these meshes.

2.1.2 The Space of Linear Polynomials

Let K be a triangle. We define the space of linear functions on K as $P_1(K)$ by

$$P_1(K) = \{v : v = c_0 + c_1x_1 + c_2x_2,\ (x_1, x_2) \in K,\ c_0, c_1, c_2 \in \mathbb{R}\}. \tag{2.3}$$

This way any function $v \in P_1(K)$ of form $v = c_0 + c_1x_1 + c_2x_2$ on K is uniquely determined by its values α_i on the three nodes of the triangle (nodal values) $\alpha_i = v(N_i),\ i = 1, 2, 3$, where $N_i = (x_1^{(i)}, x_2^{(i)})$ and we end up with a linear system

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}. \tag{2.4}$$

We can easily see by computing the system matrix's determinant that its absolute value is $2|K|$, where as $|K|$ we define the area of our triangle K, so, it being strictly positive, there is always going to be a unique solution for the system, supposing K is not degenerate. In order to use nodal values in a more handy way, meaning we won't have to solve a system for every triangle K, we can introduce a nodal basis $\{\lambda_1, \lambda_2, \lambda_3\}$, instead of our natural basis $\{1, x_1, x_2\}$, thus considering the nodal values as degrees of freedom. The elements of our new basis λ_i are defined by

$$\lambda_j(N_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \quad i, j = 1, 2, 3, \tag{2.5}$$

and using them we can express any function v of $P_1(K)$ as

$$v = \alpha_1\lambda_1 + \alpha_2\lambda_2 + \alpha_3\lambda_3, \tag{2.6}$$

where $\alpha_i = v(N_i)$.

2.1.3 The Space of Continuous Piecewise Linear Polynomials

To construct continuous piecewise linear functions on a mesh $\mathcal{K} = \{K\}$ is simple. Requiring any such function v to belong to $P_1(K)$ for each triangle K of our mesh $\mathcal{K}$, and to have continuity between neighboring triangles, we have the intended space of all continuous piecewise linear polynomials V_h, defined as

$$V_h = \{v : v \in C^0(\Omega),\ v|_K \in P_1(K),\ \forall K \in \mathcal{K}\}, \tag{2.7}$$

and by $C^0(\Omega)$ we denote the space of continuous functions on Ω.

Shall we construct a basis for V_h, we need to show, at first, that a function v in V_h is uniquely determined by its nodal values

$$\{v(N_j)\}_{j=1}^{n_p}, \tag{2.8}$$

and, conversely, that each nodal value set determines a unique function v in V_h. As stated above, nodal values determine a unique function in $P_1(K)$ for every $K \in \mathcal{K}$, so our space V_h is uniquely determined by its values on the nodes. To continue with our proof, we consider two triangles K_1 and K_2 that share an edge $E = K_1 \cap K_2$ between them. Assume v_1 and v_2 to be two linear polynomials in $P_1(K)$ and $P_2(K)$ respectively, determined uniquely by two sets of nodal values on K_1 and K_2. Since v_1 and v_2 are linear polynomials on K_1 and K_2 they are on E too, and, since they coincide on the apexes of E, we conclude that $v_1 = v_2$ on E. That means they are continuous, and

we have shown that, given any nodal values set, there is a unique continuous piecewise linear polynomial defined by them.

To define, now, our basis $\{\phi_j\}_{j=1}^{n_p}$ of V_h, we assume the nodal values to be the degrees of freedom we want, such that

$$\phi_j(N_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \quad i, j = 1, 2, \ldots, n_p. \tag{2.9}$$

If we construct a two-dimensional basis function on a general triangle mesh in a computing environment, we can see the shape of it to seem like a hat, so we refer to them as hat functions. Using the hat functions basis, we can write any and every function v in V_h as

$$v = \sum_{i=1}^{n_p} \alpha_i \phi_i, \tag{2.10}$$

where $\alpha_i = v(N_i),\ i = 1, 2, \ldots, n_p$, the nodal values of v.

2.1.4 The Space of Piecewise Constants

A piecewise constant function is the function that results from differentiating a continuous piecewise linear function, respectively, over either of the two-dimensional space coordinates x_1 or x_2. By doing this we can see, since we differentiate linear functions, that the piecewise constant functions are discontinuous.

The collection of such functions forms the space of piecewise constants W_h, defined by

$$W_h = \{w : w|_K \in P_0(K),\ \forall K \in \mathcal{K}\}, \tag{2.11}$$

where $P_0(K)$, similar to $P_1(K)$, is the space of constant functions on an element K.

At this point, we will introduce two operators, the average $\langle\cdot\rangle$ and the jump $[\cdot]$, in order to measure the size of $w \in W_h$ on the edges, where the discontinuity lies. Considering the two adjacent elements K^- and K^+ with their shared edge E and their unit normals n^- and n^+ on element boundaries ∂K^- and ∂K^+ respectively, we define the average of w on E by

$$\langle w \rangle = \frac{w^- + w^+}{2}, \quad x \in E, \tag{2.12}$$

and the jump of w on E by

$$[w] = w^+ - w^-, \quad x \in E, \tag{2.13}$$

where $w^{\pm} = w|_{K^{\pm}}$. Since n^+ and n^- point in opposite directions, we assign a new fixed normal $n_E = n^+ = -n^-$ on E so that $w^{\pm} = \lim_{\epsilon \to 0} \left[w(x \mp \epsilon n_E) \right]$, for any $x \in E$. When E lies on the boundary $\partial\Omega$ of our domain, n_E, as its normal, coincides with the boundary's normal, and we define $\langle w \rangle = w^+$ and $[w] = x^+$. These average and jump operators are used to generalize matrices $W = \{w_{ij}\}$ in W_h.

Having fully defined all the aspects of piecewise polynomials, we have the tools to approximate every function f described by a PDE upon a mesh that is made out of elements of our domain using classic interpolation methods from numerical analysis, such as linear B-Splines.

2.2 Functions Spaces

In this section, we introduce spaces in which lie the functions that are described by PDEs and used by the FEM. We will start with standard normed vector spaces and conclude with Hilbert, Banach, and L^p spaces, along with some properties of the functions in them.

2.2.1 Normed Vector Spaces

We use the basic definition of a real vector space V from the linear algebra of a set with two operations: $+ : V \times V \to V$ and $\cdot : \mathbb{R} \times V \to V$ that satisfy the basic axioms:

- $u + v = v + u, \quad \forall u, v \in V,$
- $(u + v) + w = u + (v + w), \quad \forall u, v, w \in V,$
- $\exists \mathbf{0} \in V : \mathbf{0} + u = u = u + \mathbf{0}, \quad \forall u \in V,$
- $\exists (-u) \in V : u + (-u) = \mathbf{0} = (-u) + u, \quad \forall u \in V,$
- $\lambda \cdot (u + v) = \lambda \cdot u + \lambda \cdot v, \quad \forall u, v \in V,\ \lambda \in \mathbb{R},$
- $(\lambda + \mu) \cdot u = \lambda \cdot u + \mu \cdot u, , \quad \forall u \in V,\ \lambda, \mu \in \mathbb{R},$

simplifying the notation of scalar product from $\lambda \cdot u$ to λu for all λ in $\mathbb{R}$ and u in V.

We will also use the known definition of the norm on a vector space as a mapping $\| \cdot \| : V \to \mathbb{R}$ that satisfies the properties:

- $\|u + v\| \leq \|u\| + \|v\|, \quad \forall u, v \in V,$

- $\|\lambda u\| = |\lambda|\|u\|, \quad \forall u \in V,\ \lambda \in \mathbb{R},$
- $\|u\| \geq 0,\ \forall u \in V$, with $\|u\| = 0 \Leftrightarrow u = \mathbf{0}$.

Definition 2.2.1. *A semi-norm $|\cdot| : V \to \mathbb{R}$ is a mapping that satisfies all the aforementioned requirements of a norm, except $|u| = 0 \Leftrightarrow u = \mathbf{0}$.*

We call a normed vector space the vector space equipped with a norm, and obviously since there are many norms and vector spaces, the combinations are plenty too.

A finite minimal set of vectors $\{\phi_i\}_{i=1}^n$ that span V is called a basis for the vector space V and any and every vector v in V can be written as a unique linear combination of this basis $v = c_1\phi_1 + c_2\phi_2 + \cdots + c_n\phi_n$, where $c_i,\ i = 1, 2, \ldots, n$ are coefficients in $\mathbb{R}$. A basic property of the basis of the vector space V is linear independencey, meaning that $v = \mathbf{0}$ if and only if $c_i = 0,\ \forall i$. Moreover, the dimension of V is equal to the maximum number of linearly independent vectors, or, else, the number of basis vectors n.

Definition 2.2.2. *A functional, or linear form, on a vector space V is a mapping $l(\cdot) : V \to \mathbb{R}$ that satisfies*

- $l(u + v) = l(u) + l(v),\ \forall u, v \in V,$
- $l(\lambda u) = \lambda l(u),\ \forall u \in V,\ \lambda \in \mathbb{R},$

and if there is a constant C s.t.

$$|l(u)| \leq C\|u\|,\ \forall u \in V,$$

then the linear form is continuous or bounded.

The norm of the linear form $l(\cdot)$ is defined by

$$\|l\| = \sup_{u \in V} \frac{|l(u)|}{\|u\|}, \tag{2.14}$$

and the set of all continuous linear forms on V is called dual space V^* of V.

Similarly to the linear form, we define the bilinear form

Definition 2.2.3. *A bilinear form on a vector space V is a mapping $\alpha(\cdot, \cdot) : V \times V \to \mathbb{R}$ that satisfies*

- $\alpha(u + v, w) = \alpha(u, w) + \alpha(v, w),\ \forall u, v, w \in V,$

- $\alpha(u, v + w) = \alpha(u, v) + \alpha(u, w),\ \forall u, v, w \in V,$
- $\alpha(\lambda u, v) = \lambda\alpha(u, v),\ \forall u, v \in V,\ \lambda \in \mathbb{R},$
- $\alpha(u, \lambda v) = \lambda\alpha(u, v),\ \forall u, v \in V,\ \lambda \in \mathbb{R},$

and if there is a constant C s.t.

$$|\alpha(u, v)| \leq C\|u\|\ \|v\|,\ \forall u, v \in V,$$

then the bilinear form is continuous or bounded. Also, the bilinear form is symmetric if

$$\alpha(u, v) = \alpha(v, u), \quad \forall u, v \in V.$$

Definition 2.2.4. *Inner product is called a symmetric bilinear form $\alpha(\cdot, \cdot)$ if, also,*

$$\alpha(u, u) \geq 0,\ \forall u \in V \text{ with } \alpha(u, u) = 0 \Leftrightarrow u = \mathbf{0},$$

which for short we write as $(\cdot, \cdot)$.

An inner product defines an induced norm as

$$\|u\| = (u, u), \tag{2.15}$$

on V, and that holds also for every bilinear form $\alpha(\cdot, \cdot)$.

Again, in a similar way as before, we call inner product vector space a vector space equipped with an inner product. In such spaces, various useful relations hold such as the Cauchy-Schwarz inequality

$$(u, v) \leq \|u\|\ \|v\|. \tag{2.16}$$

We define the angle θ between two vectors $u, v \in V$ by

$$\theta = \arccos \frac{(u, v)}{\|u\|\ \|v\|}, \tag{2.17}$$

and the two vectors are called orthogonal if and only if $\theta = 90°$, hence if and only if $(u, v) = 0$.

Supposing u and v are orthogonal, we have the Pythagorean Theorem that holds with

$$\|u + v\|^2 = \|u\|^2 + \|v\|^2, \tag{2.18}$$

and the Parallelogram law, too, with

$$\|u + v\|^2 + \|u - v\|^2 = 2\|u\|^2 + 2\|v\|^2, \tag{2.19}$$

for all u, v orthogonal vectors in an inner product vector space V.

2.2.2 Banach and Hilbert Spaces

In this part, we introduce Banach and Hilbert spaces because of their properties, which will provide great assistance in the use of FEM, with their most important property being completeness.

Definition 2.2.5. *Let V be a normed vector space and $\{v_i\}_{i=1}^{\infty}$ a sequence of elements $v_i \in V,\ i = 1, 2, \ldots$. We call the sequence $\{v_i\}_{i=1}^{\infty}$ Cauchy if there is a positive integer n, such that for all $\epsilon > 0$ holds*

$$\|v_i - v_j\| \leq \epsilon, \quad \textit{for} \ \ i, j \geq n. \tag{2.20}$$

Definition 2.2.6. *A sequence $\{v_i\}_{i=1}^{\infty}$ is called convergent if there exists a vector $v \in V$ such that for all $\epsilon > 0$ there is a positive integer n, such that*

$$\|v - v_i\| \leq \epsilon, \quad \textit{for } i \geq n. \tag{2.21}$$

Shall a sequence be convergent, we can easily see that it is also Cauchy. If every Cauchy sequence in a normed vector space is also convergent, then the vector space is called complete.

Definition 2.2.7. *A Banach space is a complete normed vector space.*

Definition 2.2.8. *A Hilbert space is a complete inner product vector space.*

We will see now how completeness, as a key property for the Hilbert and Banach spaces, plays an important role when proving the existence of solutions to certain problems.

The idea is that supposing we have a sequence of approximate solutions produced by an iterative method, we can show that the sequence is a Cauchy sequence, so, using the completeness of our space, the sequence converges.

We introduce some more helpful properties of Hilbert spaces before continuing on L^p spaces.

Definition 2.2.9. *A linear subspace $V_0 \subset V$ of a Hilbert space V is called closed if all sequences in V_0 have a limit in V_0, meaning that if $\{v_i\}_{i=1}^{\infty}$ with $v_i \in V_0$ and $v_i \to v$ then it is implied that $v \in V_0$. V_0 is also a Hilbert space under the same norm as V.*

Definition 2.2.10. *Supposing $(\cdot, \cdot)$ is the inner product of a Hilbert space V. The orthogonal complement of a closed linear subset V_0 of V is the set $V_0^{\perp} = \{v \in V : (v, v_0) = 0,\ \forall v_0 \in V_0\}$ and is also a closed linear subspace of V.*

Here we can define the orthogonal decomposition of V as the direct sum of the subspace V_0 and its orthogonal complement $V_0^{\perp}$ as in $V = V_0 \oplus V_0^{\perp}$.

As a consequence, we have that, for any $u \in V$, there exists a unique function $u_0 \in V_0$ that is closest to u, with respect to the induced norm $\|\cdot\|_V^2 = (\cdot,\cdot)$, so, using the norm, we can describe u_0 as the best approximation result as

$$\|u - u_0\|_V \leq \|u - v_0\|, \quad \forall v_0 \in V_0. \tag{2.22}$$

2.2.3 L^p Spaces

Following the Banach and Hilbert spaces definitions, we will now define the vector spaces of functions, which originate from a very important family of functions known as L^p spaces.

The $L^p(\Omega)$ function spaces are described as

$$L^p(\Omega) = \{v : \Omega \to \mathbb{R} \mid \|v\|_{L^p(\Omega)} < \infty\}, \tag{2.23}$$

where $\|v\|_{L^p(\Omega)}$ is defined by

$$\|v\|_{L^p(\Omega)} = \left(\int_\Omega |v|^p \, dx\right)^{1/p}, \quad 1 \leq p \leq \infty, \tag{2.24}$$

and for $p \to \infty$ we have

$$\|v\|_{L^p(\infty)} = \sup_{x\in\Omega} |v(x)|. \tag{2.25}$$

The integral above is called the Lebesgue integral and allows the integration of more general functions that do not need to have well-defined point values.

We can easily show, using the definition given in the section before, that $L^p(\Omega)$ are Banach spaces for all $1 \leq p \leq \infty$. Specifically, if $p = 2$ then we have $\|v\|_{L^2(\Omega)} = (v,v)_{L^2(\Omega)}$, where $(u,v)_{L^2(\Omega)} = \int_\Omega uv \, dx$ is the $L^2(\Omega)$ inner product, so $L^2(\Omega)$ is a Hilbert space. For other values of p, our norm $\|\cdot\|_{L^p(\Omega)}$, $p \neq 2$ is not given by any inner product, so $L^p(\Omega)$ spaces are only Banach spaces.

2.2.4 Weak Derivatives

We can see that, in Hilbert spaces, functions are not regular enough, in general, and as a result, the standard definition of the derivative does not make sense. To do so, the concept of the weak derivative is introduced.

Firstly, let $C^k(\Omega)$ be the space of all the functions in Ω, that are $k < \infty$ times continuously differentiable, and $\mathcal{D}(\Omega)$ be the space of compactly supported functions in Ω, that are infinitely differentiable with

$$\mathcal{D}(\Omega) = \{\phi \in C^\infty(\Omega) : supp(\phi) \subset\subset \Omega\}, \tag{2.26}$$

where $supp(\phi)$ is the closure of the open set $\{x \in \Omega : \phi(x) \neq 0\}$.

Definition 2.2.11. *A function ϕ is said to have compact support with respect to Ω if $supp(\phi)$ is a compact set ω that is a subset of the interior of Ω.*
A subset $\omega \in \Omega$ is compact if and only if it is closed and bounded.

With $supp(\phi) \subset\subset \Omega$ we signify that the support of ϕ is compactly contained in Ω.

This way all functions in $\mathcal{D}(\Omega)$ can be infinitely differentiated, and within Ω, they are strictly non-zero. Another very useful property is that all these derivatives become zero and vanish sufficiently close to the boundary $\partial\Omega$.

Now, should we want to differentiate a function $\phi \in \mathcal{D}(\Omega)$, we use the classical partial derivative

$$D^a\phi = \prod_{i=1}^{d} \left(\frac{\partial}{\partial x_i}\right)^{a_i} \phi, \quad \phi \in \mathcal{D}(\Omega), \tag{2.27}$$

where $a = (a_1, \ldots, a_d)$ is a d-tuple of non-negative integers a_i and its order is defined by

$$|a| = \sum_{i=1}^{d} a_i. \tag{2.28}$$

Integrating partially and using the property that the boundary terms vanish, we get that

$$\int_\Omega \frac{\partial u}{\partial x_i}\phi \, dx = -\int_\Omega u\frac{\partial \phi}{\partial x_i} \, dx, \quad \forall \phi \in \mathcal{D}(\Omega), \tag{2.29}$$

for $u \in C^1(\Omega)$. Shall we repeat the above identity, we will have that

$$\int_\Omega (D^a u)\phi \, dx = (-1)^{|a|} \int_\Omega u D^a \phi \, dx, \quad \forall \phi \in \mathcal{D}(\Omega), \tag{2.30}$$

for $u \in C^{|a|}(\Omega)$.

Definition 2.2.12. *The space of locally integrable functions is defined as*

$$L^1_{loc}(\Omega) = \{v : v \in L^1(K), \ \forall K \subset\subset \Omega\}. \tag{2.31}$$

Let u be a locally integrable function, if there is another locally integrable function g, such that

$$\int_\Omega g\phi \, dx = (-1)^{|a|} \int_\Omega u D^a \phi \, dx, \quad \forall \phi \in \mathcal{D}(\Omega), \tag{2.32}$$

then g is the weak derivative $D^a u$ of u.

Weak derivatives share many basic properties with classical derivatives, such as linearity, the chain rule, etc.

2.2.5 Sobolev Spaces

After defining weak derivatives, we introduce the concept of the Sobolev spaces that include functions with very important properties and can easily be connected with the concept of Hilbert spaces.

Definition 2.2.13. *A Sobolev space $W_k^p(\Omega)$ is defined as*

$$W_k^p(\Omega) = \{u \in L^1_{loc}(\Omega) : \|u\|_{W_k^p(\Omega)} < \infty\}, \tag{2.33}$$

where $\|u\|_{W_k^p(\Omega)}$ is the Sobolev norm of u defined by

$$\|u\|_{W_k^p(\Omega)} = \left(\sum_{|a|\leq k} \|D^a u\|^p_{L^p(\Omega)} \right)^{1/p}, \quad 1 \leq p \leq \infty, \tag{2.34}$$

and

$$\|u\|_{W_k^p(\Omega)} = \max_{|a|\leq k} \|D^a u\|_{L^\infty(\Omega)}, \quad p = \infty, \tag{2.35}$$

assuming that $u \in L^1_{loc}(\Omega)$ and that all weak derivatives $D^a u$ with $|a| \leq k$ exist, with k being a non-negative integer.

We can see that $W_k^p(\Omega)$ is an $L^p(\Omega)$ space with functions u that have weak derivatives $D^a u$, which also lie in $L^p(\Omega)$.

It can be shown that $C^\infty(\Omega)$ is dense in $W_k^p(\Omega)$ for $1 \leq p < \infty$ provided that $\partial\Omega$ is sufficiently regular, which means that smooth functions lie close to Sobolev functions. We can use sequences of smooth functions, which are simpler and easier to work with, to approximate Sobolev functions, giving us this way the advantage of avoiding using weak derivatives. These aprroximations hold within the interior of Ω and also up to the boundary given that $\partial\Omega$ has C^1 continuity. We can, alternatively, define $W_k^p(\Omega)$ as the completion of $C^\infty(\Omega)$, with respect to the abovementioned Sobolev norm $\|\cdot\|_{W_k^p(\Omega)}$, meaning that all limits of all sequences of $C^\infty(\Omega)$ that have a norm less than ∞ are contained in $W_k^p(\Omega)$. Implementing this concept, the idea is to first establish the desired results for smooth functions and then generalize these results to Sobolev functions with limits.

By the definition, we can see that Sobolev spaces are also Banach spaces for all $1 \leq p \leq \infty$, but, especially for $p = 2$, $W_k^2(\Omega)$ is also a Hilbert space. Its inner product and norm will be

$$(u, v)_{W_k^2(\Omega)} = \sum_{|a|\leq k} (D^a u, D^a v)_{L^2(\Omega)}, \tag{2.36}$$

$$\|u\|^2_{W^2_k(\Omega)} = \sum_{|a|\leq k} \|D^a u\|^2_{L^2(\Omega)}, \tag{2.37}$$

The last case, for $p = 2$, is the most common in FEM, so we use the notation of $H^k(\Omega) = W^2_k(\Omega)$ to state the basic property of it, being a Hilbert space.

The dimension d of the domain $\Omega \subset \mathbb{R}^d$ has a significant role in the regularity of functions in Hilbert spaces, as we can show, for example, that $H^1(\Omega)$ functions are continuous on $\mathbb{R}^1$, may lack values at certain isolated points on $\mathbb{R}^2$, and be discontinuous along a curve on $\mathbb{R}^3$.

2.3 Solving Large Sparse Linear Systems

As we will see in the following sections, the Finite Element Method gives rise to linear systems, in the form of $Ax = b$, which have to be solved in order for us to end up with a solution for a given problem. The size of these systems is usually large because it uses the number of nodes in the mesh that we have built. Since we have quite large numbers, with respect to the dimensions of the created matrices, we make use of methods and algorithms that help us solve the linear systems in terms of low computational complexity, time consumption, and memory requirements [7]. Problems that are easier to be dealt with, concerning the above, are sparse linear systems, which also comprise the majority of the cases. The sparsity arises from the existence of very few non-zero entries compared to their size, and this is the outcome of the fact that finite element functions, as they were defined, have limited support and only interact with their closest neighbors.

2.3.1 Projection methods

Solving problems that are more than two-dimensional can lead to the creation of linear systems that direct methods can't always be implemented on, so we use smart and fast iterative methods, which mainly focus on the residual $r = b - Ax$ being orthogonal to subspaces of $\mathbb{R}^n$ as does, in relation to the finite element method, for the residual of a partial differential equation.

We seek a solution approximation $\tilde{x}$ to $Ax = b$ from an m-dimensional subspace $K \subset \mathbb{R}^n$, in a way that the residual $r = b - A\tilde{x}$ is orthogonal to another m-dimensional subspace $L \subset \mathbb{R}^n$, such that

$$w^T r = w^T(b - A\tilde{x}) = 0, \quad \forall w \in L. \tag{2.38}$$

If we have a starting point x_0 to approximate our solution, $\tilde{x}$ must be sought in the space

$$x_0 + K. \tag{2.39}$$

So, we have that

$$\tilde{x} = x_0 + d, \tag{2.40}$$

where d is a vector in K. Then if we replace $\tilde{x}$ in (2.38) our problem becomes

$$w^T r = w^T(b - A(x_0 + d)) = w^T(r_0 - Ad) = 0, \quad \forall w \in L, \tag{2.41}$$

where $r_0 = b - Ax_0$ is our initial residual.

Let $V = [v_1, \ldots, v_m]$ and $W = [w_1, \ldots, w_m]$ are two $n \times m$ dimensional matrices with $\{v_i\}_{i=1}^m$ and $\{w_i\}_{i=1}^m$ their columns, which are also a basis for K and L, respectively. We can express d as the product Vy of our basis V and an $m \times 1$ vector y to be determined, so from (2.40) we get that

$$\tilde{x} = x_0 + Vy, \tag{2.42}$$

and from (2.41) that

$$w^T(r_0 - AVy) = 0, \quad \forall w \in L. \tag{2.43}$$

To achieve orthogonality we can replace w with W to express every $w \in L$, since W is a basis for L, and we will have that

$$W^T(r_0 - AVy) = 0, \tag{2.44}$$

or

$$W^T r_0 = W^T AVy. \tag{2.45}$$

Now, if the $W^T AV$, which is an $m \times m$ matrix, can be inverted we replace y in (2.42) and we have

$$\tilde{x} = x_0 + V(W^T AV)^{-1} W^T r_0. \tag{2.46}$$

2.3.2 Krylov Subspaces

Many very important iterative methods, that are used to produce solutions for sparse linear systems, use projection onto Krylov subspaces.

Definition 2.3.1. *The m-th Krylov subspace $K_m(A; v) \subset \mathbb{R}^n$ is defined as*

$$K_m(A; v) = span\{v, Av, A^2 v, \ldots, A^{m-1} v\}, \tag{2.47}$$

where A is a given $n \times n$ matrix and v a given $n \times 1$ vector. We say that A and v generate K_m.

The idea is that the solution $\tilde{x}$ to $Ax = b$ has a representation in a Krylov space and if a good approximation to x can be found then this will be in K_m with a space dimension m compared to matrix dimension n.

2.3.3 Conjugate Gradient

The most famous Krylov method, which is also very effective in solving linear systems, is the Conjugate Gradient (CG) method and derives from combining the prototype projection method mentioned in (2.46) with the Krylov space $K_m(A; v)$.

It can be applied in a linear system $Ax = b$, with A being a symmetric and positive definite matrix (SPD). Firstly, we consider a projection method with similar spaces $K = L = K_m(A; r_0)$, with r_0 the initial residual, and our aim is to create an orthogonal basis made of search directions p_m that, in a limited number of steps, will give us a significantly close approximation of $\tilde{x}$. We start by choosing $p_1 = r_0$ and in every step we calculate every new vector of our Krylov space $t_m = Ap_m$. Using the prototype projection type of (2.46), since $K = L = K_m(A; r_0)$, we get that $x_m = x_{m-1} + \alpha_m p_m$, where $\alpha_m = (r_{m-1}^T r_{m-1})/(p_m^T t_m)$. To find the residual in every step, we use the formula $r_m = r_{m-1} - \alpha_m t_m$ and, finally, we compute our gradients using the Gram-Schmidt algorithm in order for every new vector to be orthogonal to all previous ones, thus having $p_{m+1} = r_m + \beta_{m+1} p_m$, where β_{m+1} is the orthogonal projection coefficient $\beta_{m+1} = (r_m^T r_m)/(r_{m-1}^T r_{m-1})$.

The final form of the CG method algorithm is

Algorithm 1 Conjugate Gradient

Require: A, b, x_0
1: $r_0 \leftarrow b - Ax_0$
2: $p_1 \leftarrow r_0$
3: **for** $m = 1, 2, \ldots$ until convergence **do**
4: $\quad t_m \leftarrow Ap_m$
5: $\quad \alpha_m \leftarrow (r_{m-1}^T r_{m-1})/(p_m^T t_m)$
6: $\quad x_m \leftarrow x_{m-1} + \alpha_m p_m$
7: $\quad r_m \leftarrow r_{m-1} - \alpha_m t_m$
8: $\quad \beta_{m+1} \leftarrow (r_m^T r_m)/(r_{m-1}^T r_{m-1})$
9: $\quad p_{m+1} \leftarrow r_m + \beta_{m+1} p_m$
10: **end for**

Normally, CG should converge on to the solution after at most n iterations, where n is also the dimension of $\mathbb{R}^n$, but, since there is finite precision of computer arithmetic, this is not what actually happens.

The CG method converges with a rate that primarily depends on the condition number $\kappa(A)$ of A. We use the error $e_m = x - x_m$ where x_m results

from the m-th iteration and the A-norm $\|v\|_A = \sqrt{v^T A v}$, so that we have

$$\|e_m\| \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^m \|e_0\|. \tag{2.48}$$

2.3.4 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization method, inspired by Gradient methods, frequently used in cases of machine learning examples and, less frequently for the solution of PDEs and the minimization of functions. As far as PDE solving is involved, it uses the initial residual vector $r^{(0)}$, more specifically, a sample of its dimensions, and forms a new residual vector $r_i^{(0)}$, with all its components equal to zero except those that were sampled from $r^{(0)}$, that is used to find the optimal values of the function f, in order to minimize it. To do that, it makes use of the iterative recalculation of the residual vector $r^{(k)}$, a procedure that locates critical points of the correspondent quadratic function that is created.

Considering, now, the matrix A, which is created either by the quadratic function $u(x) = \frac{1}{2} x^T A x + b$, where A symmetric positive-definite, or by the bilinear form of the PDE, we seek a multi-dimensional function u, where the residual r is minimized, using a sampled part of the residual $r_{i_k}^{(k)}$, and the iterative relation

$$u^{(k+1)} = u^{(k)} - A r_{i_k}^{(k)}, \tag{2.49}$$

or

In (2.49), i_k is an index sampled, at each step, from a uniform distribution

$$i_k \sim \mathcal{U}\{1, 2, \ldots, N\}, \tag{2.50}$$

and, as we can see, the SGD method uses only one sample at a time.

In cases of deterministic problems, where the solution of a PDE or the minimization point of a function u is specific, simple gradient methods should be preferred, instead of stochastic ones since the sampling of components from the residuals can prove to be, only partially, optimizing the solution to the problem, while the complete vector, even though it comes with a higher computational cost, improves drastically faster the solution achievement. Although, in cases of Machine Learning problems, due to the existence of an enormous introductory dataset, that needs to be filtered for the proper identification of new data inserted in the problem, stochastic optimization methods, such as SGD, are optimal.

The SGD method algorithm, as created for the solution of a PDE, is the following:

Algorithm 2 Stochastic Gradient Descent

Require: A, b, f ▷ We use the quadratic form of the PDE.

$r^{(0)} = b$ ▷ Initial residual with $\dim r = \dim f$.

$u^{(0)} \leftarrow 0$

for $k = 1, 2, \ldots,$ until convergence **do**

  sample uniformly $i_k \in \{1, \ldots, \dim r\}$

  $a_k = \dfrac{\langle r_{i_k}^{(k)}, r_{i_k}^{(k)} \rangle}{\langle r_{i_k}^{(k)}, r_{i_k}^{(k)} \rangle_A}$

  $u^{(k+1)} = u^{(k)} + a_k r_{i_k}^{(k)}$

  $r^{(k+1)} = r^{(k)} - a_k A r_{i_k}^{(k)}$

end for

Through the years and until today, several extensions of the SGD have been formulated, such as minibatch, momentum, AdaGrad, Adam, etc., and are primarily used in cases of Deep Learning problems [7].

2.4 Finite Element Analysis

Here, we briefly go back to some very important and frequently used properties of integration in $\mathbb{R}^2$. We continue with the classical definition of a finite element, making use of polygons, polynomial spaces, and the functionals we talked about in previous sections, as we also introduce the Lax-Milgram Lemma, a fundamental theorem for FEM. We close both the section and the chapter, with a detailed study of the Finite Element Approximation, considering the general Finite Element Method, the construction of Stiffness Matrices and Load Vectors, and, finally, the Gallerkin Orthogonality [4].

2.4.1 Green's Formula

Let our domain be Ω in $\mathbb{R}^2$, its boundary $\partial\Omega$, and an exterior unit normal u. The first equation we recall is a form of the divergence theorem

$$\int_\Omega \frac{\partial f}{\partial x_i}\, dx = \int_{\partial\Omega} f n_i\, ds, \quad i = 1, 2, \tag{2.51}$$

with n_i being the i-th component of n.

Now, shall we set $f = fg$ we will have

$$\int_\Omega \frac{\partial f}{\partial x_i} g \, dx = -\int_\Omega f \frac{\partial g}{\partial x_i} \, dx + \int_{\partial\Omega} f g n_i \, ds, \quad i = 1, 2. \tag{2.52}$$

If we apply $f = w_i$ and $g = v$ in the above integral, where w_i are the components of a vector field w on Ω, we obtain

$$\int_\Omega (\nabla \cdot w)\, v \, dx = -\int_\Omega w \cdot \nabla \, v \, dx + \int_{\partial\Omega} (w \cdot v) n \, ds, \tag{2.53}$$

and finally to get Green's Formula we choose $w = -\nabla u$ so we have

$$\int_\Omega -\Delta u v \, dx = \int_\Omega \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega} n \cdot \nabla u v \, ds. \tag{2.54}$$

2.4.2 The Finite Element

To formally define a finite element we need:

- A polygon $K \subset \mathbb{R}^d$,
- A polynomial function space P on K,
- A set of linear functionals $L_i(\cdot), i = 1, 2, \ldots, n$ where $n = \dim(P)$ which will be our degrees of freedom.

We will mainly focus on the two-dimensional space, so we have that $d = 2$, and our polygons K lie on $\mathbb{R}^2$, with their most common type being a triangle. Each polygon K stems from a mesh $\mathcal{K} = \{K\}$ of our computational domain Ω. Triangle meshes are easier to represent domains with curved boundaries, while using other shapes of $\mathbb{R}^2$, such as rectangular or square meshes, may be more difficult to use but simpler to implement in a computing environment.

Firstly, let us equip our polynomial function space P with the basis $\{S_j\}_{j=1}^n$ of the so-called, shape functions S_j.

Definition 2.4.1. *A finite element is said to be unisolvent if the functionals $L_i(\cdot)$ can uniquely determine the shape functions.*

Unisolvency is a very important property for finite elements as it acts as a necessary compatibility condition for $L_i(\cdot)$, P, and K. By the definition above, we have that $L_i(v) = 0 \equiv v = 0$ for all $v \in P$ and all i. We can easily calculate these shape functions S_j by solving the linear system

$$L_i(S_j) = \delta_{ij}, \quad i, j = 1, 2, \ldots, n, \tag{2.55}$$

where δ_{ij} is the symbol of Kronecker. Taking linear combinations of shape functions and coefficients leads to getting polynomial or finite element functions in P on each polygon K.

Having specified the shape functions on each polygon K, we need to take into account the behavior of these shape functions between adjacent polygons. Depending on the shape functions we choose, the functionals determine both local and global properties of our finite element space V_h. We end up having families of finite elements that share similar properties, such as continuity, although they might be different in other aspects, such as polynomial order. The most popular such family is considered to be the Lagrange family with the defining functionals being

$$L_i(v) = v(N_i), \quad i = 1, 2, \ldots, n, \tag{2.56}$$

where N_i are the selected node points that are n in number. Each of the functionals consists of a point evaluation of v, making them as simple as they can be. The resulting set of shape functions gets to be called a nodal basis. For a two-dimensional space of $\mathbb{R}^2$ with $P = P_1(K)$ the space of linear polynomials on K triangles, the node points are the triangle vertices and the S_j shape functions are the aforementioned hat functions.

Lagrange finite elements are C^0 continuous elements, meaning they are generally continuous but have discontinuous derivatives across element boundaries, making them sufficient to approximate functions in H^1. However, in need to solve higher-order ODEs and PDEs, such as $\Delta u^2 = 0$, we need element types with higher continuity properties. Some of these are the Argyris elements, the Hermite elements, and the Morley elements, which are more elaborate in construction, using more defining functionals than Lagrange, thus making them more difficult to use.

2.4.3 The Lax-Milgram Lemma

The Lax-Milgram Lemma [8] is a fundamental theorem for the implementation of the Finite Element Method in the abstract variational problem that is stated below, and is considered the extension of the Riesz representation theorem to non-symmetric bilinear forms $\alpha(\cdot, \cdot)$.

Theorem 2.4.1 (**Lax-Milgram Lemma**). *Assume V to be a Hilbert space with inner product $(\cdot, \cdot)$, $\alpha(\cdot, \cdot)$ to be a coercive, continuous, bilinear form on V, and $l(\cdot)$ to be a continuous, linear form on V. There exists a unique solution $u \in V$ to the abstract variational problem*

$$\alpha(u, v) = l(v), \quad \forall v \in V. \tag{2.57}$$

The proof can be found in detail on [4].

2.4.4 Abstract Finite Element Method

As we know from the Lax-Milgram Theorem, the solution u to the equation (2.57) exists and is unique. Implementing the Finite Element Method means we, first, have to create the equivalent to the (2.57) problem. Let us have $V_h \subset V$ a finite-dimensional subspace of V, that consists of continuous piecewise polynomials on a mesh $\mathcal{K}$ of Ω with mesh size h. This way, we have the problem:

$$\text{Find} \quad u_h \in V_h \quad \text{such that} \quad \alpha(u_h, v) = l(v), \quad \forall v \in V_h. \tag{2.58}$$

2.4.5 Stiffness Matrix and Load Vector

Since we want to find a $u_h \in V_h$ so that (2.58) holds for every $v \in V_h$, we can substitute v with a basis $\{\phi_i\}_{i=1}^n$ for V_h, and (2.58) will become

$$\alpha(u_h, \phi_i) = l(\phi_i), \quad i = 1, 2, \ldots, n. \tag{2.59}$$

This way (2.59) holds for every linear combination of our basis functions $\phi_i, i = 1, 2, \ldots, n$, so it is equivalent to (2.58).

At this point, we make use of the Galerkin method.

Definition 2.4.2. *The Galerkin method states that the approximate solution of (2.59) lies in the subspace V_h, and is defined by*

$$u_h = \sum_{i=1}^{n} \xi_j \phi_j, \tag{2.60}$$

where ξ_j are unknown coefficients that will be determined in the next steps.

Now using the Galerkin method on (2.59) we will have

$$b_i = l(\phi_i) = \sum_{j=1}^{n} \xi_j \alpha(\phi_j, \phi_i) = \sum_{j=1}^{n} A_{ij} \xi_j, \quad i = 1, 2, \ldots, n, \tag{2.61}$$

and we have constructed an equivalent problem with n linear algebraic equations for the $\xi_j, j = 1, 2, \ldots, n$, which can be also written as

$$A\xi = b. \tag{2.62}$$

Here, A is a $n \times n$ matrix given by $A_{ij} = \alpha(\phi_j, \phi_i)$, and b is the $n \times 1$ vector given by $b_i = l(\phi_i)$. A is called the Stiffness matrix, b the load vector, and by solving the linear system $A\xi = b$ for the vector ξ we get the Finite Element solution $u_h = \sum_{j=1}^{n} \xi_j \phi_j$.

2.4.6 Implementation on Boundary-Value Problems

A Diffusion-Reaction Equation

In order to give a more specific way to execute the Finite Elements Method, we inspire from [65] and we introduce a diffusion-reaction equation, upon the solution of which we will implement a conjugate gradient algorithm. We have the following boundary-value problem

$$-\Delta u(x) + 10u(x) = f(x), \qquad x \in \Omega, \tag{2.63}$$

$$u(x) = u_D(x), \qquad x \in \partial\Omega, \tag{2.64}$$

where x is a vector in Ω.

Here, we have $u(x) = u$ the unknown function, $f(x) = f$ a given function, Δ the Laplace operator, $\Omega \subset \mathbb{R}^2$ our spatial domain, and $\partial\Omega$ the boundary of the aforementioned domain. The unknown function u has two variables $u = u(x, y)$ since we are on $\mathbb{R}^2$. The boundary condition is also known as a Dirichlet boundary condition where $u = u_D$ on $\partial\Omega$ with u_D being a given function too.

We start by multiplying with a test function $v \in \hat{V}$, and integrating the equation (2.63) over Ω, obtaining this way the so-called weak form

$$\int_\Omega (-\Delta u + 10u)v \; dx = \int_\Omega fv \; dx, \tag{2.65}$$

and, by solving this well-defined problem, we will uniquely determine the solution u that lies in a, possibly different, function space V which is called test space.

We define the trial and test spaces V and $\hat{V}$ as

$$V = \{v \in H^1(\Omega) : v = u_D \text{ on } \partial\Omega\}, \tag{2.66}$$

$$\hat{V} = \{v \in H^1(\Omega) : v = 0 \text{ on } \partial\Omega\}. \tag{2.67}$$

Then, by using the Green's Formula, we have that

$$\int_\Omega (-\Delta u + 10u)v \; dx = \int_\Omega (\nabla u \cdot \nabla v + 10uv) \; dx - \int_{\partial\Omega} n \cdot (\nabla uv - 10uv) \; ds, \tag{2.68}$$

and, since $v \in \hat{V}$, that means that $v = 0$ on the whole boundary $\partial\Omega$, and the second integral will be equal to zero. This way, the (2.65) becomes

$$\int_\Omega (\nabla u \cdot \nabla v + 10uv) \; dx = \int_\Omega fv \; dx. \tag{2.69}$$

Considering the first more general form of variational problems we stated in (2.58), here in this diffusion-reaction equation we have that

$$\alpha(u, v) = \int_\Omega (\nabla u \cdot \nabla v + 10uv)\ dx, \tag{2.70}$$

and, since $f(x, y) = xy$,

$$l(v) = \int_\Omega fv\ dx = \int_\Omega xyv\ dx. \tag{2.71}$$

The variational problem (2.69) is a continuous problem, meaning that u is defined in the infinite-dimensional space V. To find properly the approximate solution for the diffusion-reaction equation, we replace V and $\hat{V}$ with discrete (finite-dimensional) trial and test spaces, $V_h \subset V$ and $\hat{V}_h \subset \hat{V}$, thus creating the discrete variational problem

$$\text{Find } u_h \in V_h \subset V \text{ such that}$$
$$\int_\Omega (\nabla u_h \cdot \nabla v + 10u_h v)\ dx = \int_\Omega fv\ dx. \tag{2.72}$$

In this case, our domain Ω is the mesh we create, a square with sides of 1 cm that are also the borders of the domain $\partial\Omega$, stated as Dirichlet boundaries, with a global mesh size of 0.04. As the given function, we choose $f(x, y) = xy$ which lies on the H^1 function space defined on our mesh.

At this point, we use a finite set of basis functions, $\{\phi_i\}$ and $\{\hat{\phi}_i\}$, for the function spaces, V and $\hat{V}$ respectively, to describe the functions u_h and v_h as the span of the basis that spans each space. This way, we write $u_h = \sum U_i\phi_i$ and, instead of v, we will set a basis function $\hat{\phi}_i$ so that we have

$$\sum U_j \int_\Omega (\nabla\phi_j \cdot \nabla\hat{\phi}_i + 10\phi_j\hat{\phi}_i)\ dx = \int_\Omega f\hat{\phi}_i\ dx, \tag{2.73}$$

creating a linear system $AU = b$, where $A_{ij} = \int_\Omega(\nabla\phi_j \cdot \nabla\hat{\phi}_i + 10\phi_j\hat{\phi}_i)$ as our stiffness matrix, and $b_i = \int_\Omega f\hat{\phi}_i$ as the load vector. Now, we can use, according to the form of the matrix A, the proper method to solve the system and extract the approximation of the function u. In our example, we make use of the conjugate gradient method, in order to avoid calculating the inverse matrix, and, finally, approximate the solution function.

Case	Variables	L^2 error norm	Solution time (s)
FEM	$u(x)$	$2.46E-2$	0.1

Table 2.1: L^2 error production and computational cost for solving the Diffusion-Reaction equation using FEM.

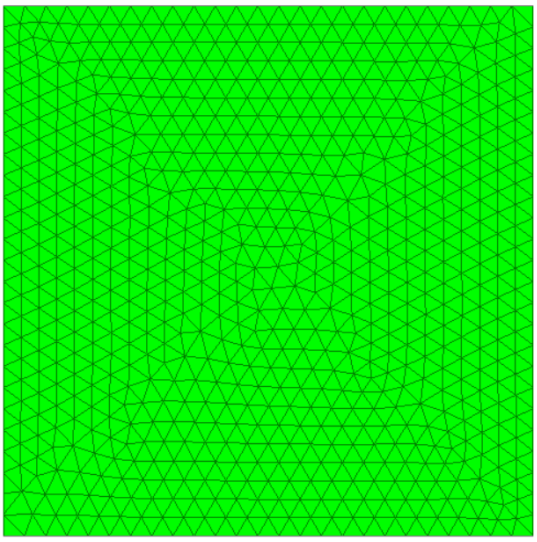


Figure 2.1: Mesh construction for the Diffusion-Reaction equation.

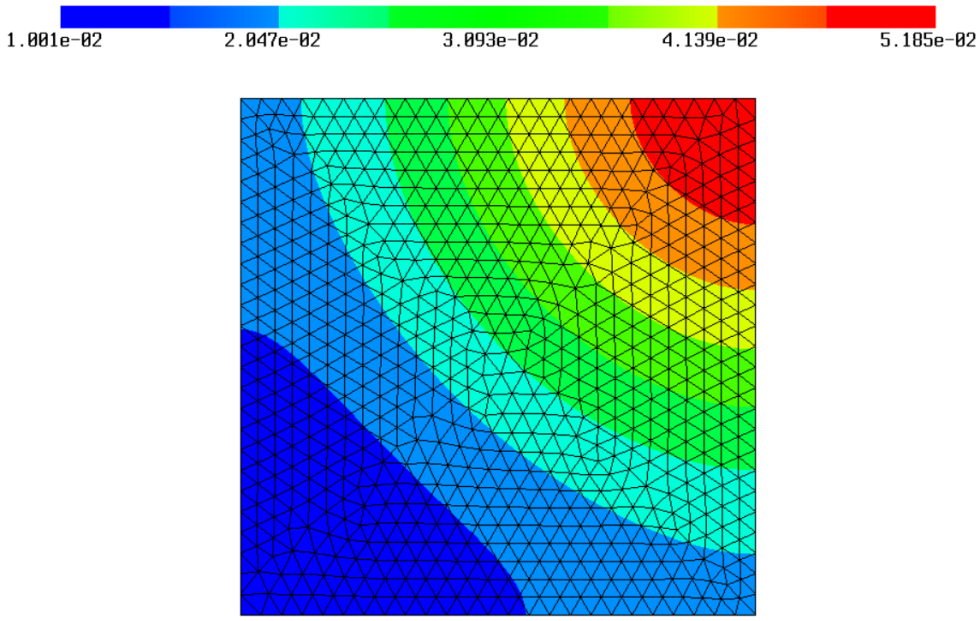


Figure 2.2: Solution of the Diffusion-Reaction equation.

Chapter 3

Reduced Basis Methods

The Reduced Basis Methods (RBMs) are a family of Reduced Order Method strategies that simplify parametrized Partial Differential Equation (PDE) problems while maintaining data integrity and result accuracy [11]. The essential idea of RBMs is to decouple the generation and projection stages of the approximation process, allowing PDEs to be solved rapidly and effectively. The RBMs, which are most commonly used in 3D systems, deliver significant computational savings, particularly in real-time studies or multi-query applications such as sensitivity analysis and optimization, when several solutions for different parameters are required [12].

Definition 3.0.1. *A Reduced Basis Method is also defined as a Galerkin projection, as stated in (2.4.2), on a Reduced Basis space, which is a low dimensional space. It is used to approximate high-fidelity solutions of equations, such as the solutions we got from applying the FEM in the previous chapter, within a given parameter space [7].*

To create a RBM, we follow a procedure, similar to the FEM, where we start with setting the parameters of the PDE and their range. These parameters represent properties, boundary conditions, geometry, etc., and we set them as the vector

$$\theta = [\theta_1, \ldots, \theta_p]^T \in \Theta, \tag{3.1}$$

where $\Theta \subset \mathbb{R}^p$ is the parameter space. We, then, define the strong form of the PDE problem based on the parameterization by

$$\text{Find } u(\theta) \in V \text{ such that: } \mathcal{L}(\theta)u(\theta) = f(\theta) \text{ in } V', \tag{3.2}$$

where

- $\Omega \subset \mathbb{R}^p$ is our computational domain,

- $V = V(\Omega)$ is a Hilbert space with functions and their derivatives that are L_2-bounded,
- V' is the dual space of V,
- $\mathcal{L} : V \to V'$ is a second-order differential operator,
- $f : V \to \mathbb{R}$ is a linear continuous form.

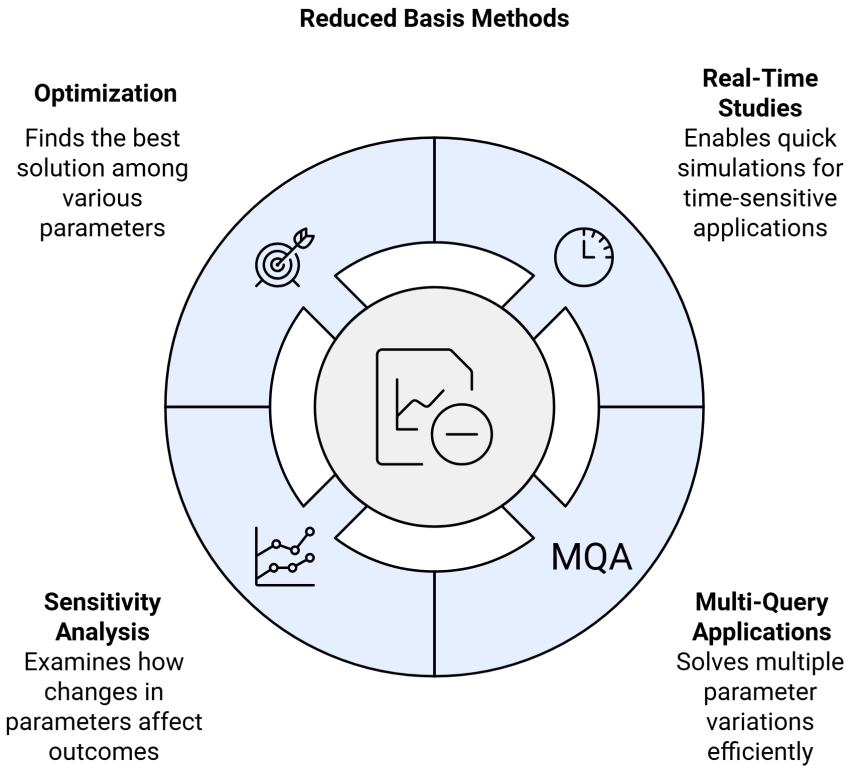


Figure 3.1: The complete concept of Reduced Basis methods.

At this point, we introduce the weak form of the previous problem, just as in FEM,

$$\text{Given } \theta \in \Theta \text{ find } u \in V \text{ such that:}$$
$$\alpha(u(\theta), v; \theta) = f(v; \theta), \quad \forall v \in V, \tag{3.3}$$

and then substitute u and v with their high-fidelity approximations u_h and v_h respectively, so the problem (3.3) becomes

$$\text{Given } \theta \in \Theta \text{ find } u_h \in V_h \text{ such that:}$$

$$\alpha(u_h(\theta), v_h; \theta) = f(v_h; \theta), \quad \forall v_h \in V, \tag{3.4}$$

for which we know that the existence and uniqueness of the solution is inherited from the weak form.

This general and abstract formulation gives us a solid theoretical foundation for the construction of RBM techniques, and, finally, a reliable and stable methodology and solution for reduced problems with controlled errors. The high computational cost of (3.4) leads us to the construction of a reduced space $V_r \subset V_h$ with $\dim(V_r) \ll \dim(V_h)$, that contains an approximate solution of (3.4). To do this, we start with the computation of a collection of solutions with high fidelity called snapshots

$$S(u, \theta) = \{u_h(\theta^1), \ldots, u_h(\theta^n)\}. \tag{3.5}$$

Then, we continue with the construction of an orthonormal basis

$$\{\zeta_1, \ldots, \zeta_r\}, \tag{3.6}$$

which is called the reduced basis, such that

$$\langle \zeta_i, \zeta_j \rangle = \delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j, \end{cases} \quad \text{for } 1 \leq i, j \leq r, \tag{3.7}$$

where r is determined by an optimality criterion that is suitable for the problem. As a last step, we compute the RB solution by

$$u_r(\theta) = \sum_{i=1}^{r} u_r^{(i)}(\theta)\zeta_i, \tag{3.8}$$

by solving the reduced problem, in order to find also the values of the unknown coefficients $u_r^{(i)}(\theta)$.

We get the reduced problem by imposing the orthogonality constraint on the residual which is given by

$$e(\theta) = f(\theta) - \mathcal{L}(\theta)u(\theta), \tag{3.9}$$

and we have

$$\text{Find } u_r(\theta) \in V_r \text{ such that:}$$
$$\langle f(\theta) - \mathcal{L}(\theta)u_r(\theta), v_r \rangle = 0, \quad \forall v_r \in V_r. \tag{3.10}$$

From (3.10) we get the solution with the smallest possible approximation error, due to the orthogonality condition which helps us find a vector u_r that is orthogonal to our reduced space V_r, and according to the Galerkin theory, solving the (3.10) problem is equivalent to solving the (3.4) problem.

Finally, we make use of the FEM procedure to discretize the above reduced problem and create a sparse linear system

$$Au = f, \tag{3.11}$$

that can be solved by simple explicit or implicit ways of linear algebra and numerical analysis respectively.

However, there are some disadvantages associated with RBMs. Creating an accurate reduced basis space needs solving the governing equations for a representative set of parameter values during the offline phase. This process can be computationally expensive, limiting the use of RBs in situations where the offline phase is resource-intensive. Furthermore, the method may encounter difficulties when dealing with nonlinearities or problems involving rapidly changing parameter dependence.

Examples of implementations of RBMs can be found in [13], [14].

3.1 Singular Value Decomposition

The Singular Value Decomposition (SVD) is a technique frequently used in some cases of RBMs and constitutes a crucial part of others. SVD is a method that has a wide range of applications, including finding solutions to inverse problems, building reduced-order models, and obtaining a low-rank approximation to a huge matrix A. This approach is useful for dealing with high-dimensional datasets and matrices because it expresses them using a small number of statistically significant directions that capture a considerable fraction of the variance. This allows us to minimize the dimension of complicated issues and systems that frequently generate data in the form of big matrices with primarily low rank, and approach them using low-rank approximations.

Suppose we have a random $m \times n$ complex matrix X large enough, we define its singular values σ_i as the real, non-negative solutions to the system

$$Xv = \sigma u, \tag{3.12}$$

$$X^H u = \sigma v, \tag{3.13}$$

where $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ are non-zero unit vectors, and as X^H we denote the Hermitian, or conjugate transpose, of X. Now, let $U = [u_1 \mid \ldots \mid u_m]$ and $V = [v_1 \mid \ldots \mid v_n]$ be an $m \times m$ and an $n \times n$ orthogonal matrix, respectively, as well as, Σ an $m \times n$ matrix with

$$\sigma_{ij} = \begin{cases} 0, & \text{for } i \neq j, \\ \sigma_i, & \text{for } i = j, \end{cases} \tag{3.14}$$

where σ_i are the singular values of X.

Definition 3.1.1. *Let X be a large $m \times n$ matrix with complex values. The SVD factorization of X is given by*

$$X = U\Sigma V^H. \tag{3.15}$$

We can observe that the singular values of the matrix X are the square roots of the eigenvalues of the square matrix XX^T, or, in other words, that the squares of the singular values of X are the eigenvalues of XX^T.

In case we have more rows than columns $m > n$ in Σ, then the matrix will have at most n non-zero elements on its main diagonal and is expressed as

$$\Sigma = \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix}, \tag{3.16}$$

where $\hat{\Sigma}$ is of $n \times n$ dimension.

Definition 3.1.2. *The reduced, or economy-sized SVD is defined as*

$$X = \hat{U}\hat{\Sigma}V^H, \tag{3.17}$$

where the columns of the matrix $\hat{U}$ correspond to the singular values of $\hat{\Sigma}$.

The coding needed to compute the SVD of a certain matrix exists in several, if not all, languages and such examples can be found in [7], while to take it even further, updates of the SVD, that can, for example, identify general colored, cross-correlated noise, instead of white noise can be found in [10].

3.2 Proper Orthogonal Decomposition

As we said before, POD and greedy algorithms are the two principal techniques that are used to create reduced spaces, and they differ in the way they construct the reduced basis stated in (3.6) [15]. POD involves identifying a set of basis functions that accurately represent the system's behavior. The basis functions are orthogonal and derived from a set of system snapshots which are typically taken by sampling the system's response at discrete time intervals [16], [17].

The SVD has a very helpful property of providing us with a low-rank approximation of the original matrix A with rank $r < \min(m, n)$, where m and n the dimensions of A. Since we can create an approximation of rank r just by retaining the first r singular values and corresponding vectors, a hierarchy of matrix approximations is formed, resumed better in the following theorem.

Theorem 3.2.1. *If $A \in \mathbb{R}^{m\times n}$ our rank-r matrix then the reduced matrices*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \quad 0 \leq k \leq r, \tag{3.18}$$

where u_i and v_i, the corresponding vectors of matrices U and V in (3.15), satisfy the optimal approximation property

$$\begin{aligned} \|A - A_k\|_F &= \min_{B\in\mathbb{R}^{m\times n},\ rank(B)\leq k} \|A - A_k\|_F \\ &= \left[\sum_{i=k+1}^{r} \sigma_i^2\right]^{1/2}, \end{aligned} \tag{3.19}$$

where $\|\cdot\|_F$ represents the Frobenius norm.

So we see that the square root of the sum of the neglected singular values, or, if we replace the Frobenius norm with a 2-norm, the dominant neglected singular value, is equal to the error of our approximation.

Now, assuming that we have a quantity of n_s high-fidelity solutions, one for each parameter sample, then the matrix S formed from their snapshots will be

$$S = [s_1|\ldots|s_{n_s}], \tag{3.20}$$

where $s_i \in \mathbb{R}^{N_h}$, and the n_s parameter samples are given by the finite sets $\{\theta^1, \ldots, \theta^{n_s}\}$ and their corresponding high-fidelity solutions s_i being

$$\{u_h(\theta^1), \ldots, u_h(\theta^{n_s})\}. \tag{3.21}$$

At this point, we introduce the SVD decomposition of the matrix S above as

$$S = U\Sigma V^T, \tag{3.22}$$

where, now,

$$\begin{aligned} U &= [u_1 \mid \ldots \mid u_{N_h}] \ \in \ \mathbb{R}^{N_h\times N_h}, \\ V &= [v_1 \mid \ldots \mid v_{n_s}] \ \in \ \mathbb{R}^{n_s\times n_s}, \\ \Sigma &= diag(\sigma_1, \ldots, \sigma_r) \ \in \ \mathbb{R}^{N_h\times n_s}, \end{aligned}$$

and the rank of the matrix Σ is

$$r \leq \min(N_h, n_s). \tag{3.23}$$

Lastly, thanks to the theorem (3.2.1), we can deduce that the error produced from the production of the minimal approximation is, indeed, optimal

so the POD basis produced is also optimal, and to further reduce the dimension, we can choose a minimal POD dimension $N_r \leq r$ as the smallest integer that satisfies

$$I(N_r) := \frac{\sum_{i=1}^{N_r} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq 1 - \epsilon^2, \tag{3.24}$$

where ϵ is the error tolerance we desire. This way, we created a basis for a reduced-order model that captures in great accuracy the overall behavior of the system [18].

But POD is not without its drawbacks. First of all, its applicability in systems with significant non-linearities may be limited because it depends on the assumption that the data is linear. Furthermore, the assumption that modes are orthogonal to one another is made by the orthogonality constraint in POD, which might not always be accurate and could limit the technique's accuracy producing less-than-ideal results. The sensitivity of POD results to initial conditions is another disadvantage. The starting point may introduce variability and impact the robustness of the analysis by influencing the decomposition process's outcome. Even with these drawbacks, Proper Orthogonal Decomposition is still a useful technique for reducing dimensionality and analyzing complex systems, especially when used carefully and taking into account its underlying presumptions.

3.3 Greedy algorithms

The different approach of greedy algorithms for the creation of a reduced basis is practically the usage of an a posteriori error estimate, on which the optimality criterion is based, in each iteration of an iterative sampling from the parameter space. This helps in cases when the computation of the eigenvalues of the POD decomposed matrix is too costly, or when lots of snapshots are required.

In any greedy optimization method that is used to construct a reduced subspace, each element is produced iteratively by adding a new basis vector at each step, instead of optimizing over all possible N-dimensional subspaces, as we did in POD in (3.2.1). This way, we start with a single sample set and its corresponding snapshot, and continue with adding additional orthonormal directions and their snapshots, based on the optimizing criterion which is the a posteriori error estimation that suits us better. Here, the evaluation of the error estimation can require many computations of high-fidelity solutions.

3.4 Proper Generalized Decomposition

In the fields of engineering and mechanics, Proper Generalized Decomposition (PGD) is a Reduced-Order Method that is frequently employed. PGD can be considered a Reduced Basis Method, although most of the time, depending on the execution, due to it having its own distinct characteristics, is classified as a separate technique belonging in the broader spectrum of Reduced-Order Methods. By breaking down a high-dimensional system into a collection of low-dimensional modes, a reduced-order model that faithfully depicts the behavior of the system can be created [19]. The foundation of PGD is the notion that a high-dimensional problem's solution can be expressed as a product of low-dimensional functions. The high-dimensional solution is projected onto a collection of basis functions to produce these functions [5]. PGD permits the use of non-orthogonal basis functions, in contrast to other ROMs [20].

PGD is a method that depends on tensors in order to approximate functions that belong in high-dimensional spaces utilizing low-dimensional subspaces. Let $u(x)$ be a function that lies within a space of high dimension and $x \in \mathbb{R}^n$ a vector. To find the corresponding subspace we start by expressing $u(x)$ as a tensor product in the form of:

$$u(x) = \sum_{i_1=1}^{r_1} \cdots \sum_{i_n=1}^{r_n} U_{i_1 \ldots i_n} f_{i_1}(x_1) \ldots f_{i_n}(x_n), \tag{3.25}$$

where all of the functions are one-dimensional, depending only on the corresponding variable and $U_{i_1 \ldots i_n}$ the coefficients that are used for the PGD approximation. The solution of a system that comprises these functions along with the coefficients and the independent variables gives us the values of $f_{i_k}(x_k)$.

PGD's capacity to manage problems with numerous dimensions is one of its main advantages. PGD is a useful substitute in situations where traditional numerical techniques frequently falter due to the "curse of dimensionality", while it can effectively lower computational complexity by breaking down the solution into lower-dimensional components, which makes it ideal for problems with particular structural features [21]. PGD also has benefits in terms of parallelization and efficiency. Large-scale problem computation times can be greatly decreased by implementing it on high-performance computing systems, while, also, this method uses less memory than conventional approaches, which is especially helpful when working with large datasets.

However, PGD has drawbacks as well, as its effectiveness depends on the particulars of the problem, and its applicability is restricted. A thorough

grasp of the physics or mathematics underlying the problem is necessary to build an accurate PGD model, and convergence problems can present difficulties that call for careful parameter tuning. Another significant drawback of PGD is its reliance on training data, while for the high-dimensional solution space to be accurately represented, there must be enough training data, which can be difficult to come by in some circumstances. Lastly, PGD might not adjust well to modifications in the underlying problem structure, needing adaptation or retraining in the event that substantial changes take place.

Chapter 4

Digital Twins

The Digital Twin (DT) is a relatively new digital technology that was initially created to predict the future behavior of systems, such as fuel consumption by a system after a prescribed time. After achieving great popularity for the last decade, DTs these days have evolved into virtual replicas of physical counterparts, by exchanging in both directions information with them. They replicate intended or actual real-world products, systems, and processes and include every aspect of their construction such as their planning, their design, their construction, the way they operate, and the way they are maintained.

To create such a system and produce predictive data from it, that would represent the actual behavior of the real-world procedure, a synergy is required between

- data that are collected, either by measurements or by observations and
- models that can describe adequately the underlying domain and physics of the system.

The challenges of the DTs are the purely data-driven perspective, because of the multiple scales and layers of complexity as far as the physics of the phenomena are concerned, and the purely model-driven approach, due to the high dimension of several uncertain parameters that might exist and are partly or completely unknown to us.

To be able to construct a DT, according to [7], we must first take into account all of the available resources such as

- the availability of vast quantities of real-time data, due to the wide use of sensors or records that exist,
- the easy accessibility to that data and their resources,

- the usage of AI, which can significantly close the gap created by the lack of knowledge on each system,
- the volume of mathematical models that exist and their ability to describe a large number of phenomena,
- the low potential cost for devices, which may be needed for computational purposes.

This way we can create a system that will accompany its physical counterpart throughout its life and identify every static and dynamic part of it, such as the initial model, the processes, the specifications in the design, or even the simulation process along with large volumes of data acquisition.

Consequently, the basic construction of a DT consists of

1. the formulation of the actual problem that governs the system,
2. the procedure of data assimilation based on the existing parameters, using variational or statistical approaches,
3. the implementation of order reduction methods, so as to extract an easier problem to solve concerning time and complexity,
4. the scientific machine learning, that is required to identify aspects of the problems that mathematical problems lack the ability to,

and, during these steps, we will come across some of the methods stated in the previous chapters that are used directly in the methodology of the assembling of a DT.

4.1 Inverse Problems

As stated in [25], if the formulation of one problem includes all or a portion of the solution of the other, then the two problems are inverse to one another and are called the direct and the inverse, exchanging information between one another. The direct problem is, most of the time, quicker and easier to solve due to the knowledge we have of the model parameters and we simply simulate the solutions. The creation of the inverse problem, though, is a much more challenging process. To effectively formulate it, we need to compare simulations and observations, in order to find the model parameters, which we solve again expecting to find results similar to the starting simulations. We can come across several inverse problems in science or engineering, where observations are common, but the phenomenon in effect may be unknown. In this part, we will focus mainly on deterministic inverse problems, although

the more realistic cases involve the construction and solution of statistical inverse problems, that can be treated by Bayesian approaches, weighted least squares and maximum likelihood, etc. For an overall review of the inverse problems [26] is recommended, as well as [27], [28] for a more computational point of view, and [29] for a mathematical approach.

We start by giving a standard and general definition of a well-posed inverse problem [30]

Definition 4.1.1. *Supposing X and Y two normed spaces, with a linear or non-linear mapping $K : X \to Y$ that connects them. A problem of determinining the value of x given the value of y such that*

$$Kx = y, \tag{4.1}$$

is well-posed, if the following three properties apply to the solution of the problem:

1. *Existence or that $\forall y \in Y$ there exists at least a $x \in X$ that solves $Kx = y$,*
2. *Uniqueness or that $\forall y \in Y$ there exists at most one $x \in X$ that solves $Kx = y$,*
3. *Stability or that x as the solution depends continuously on the given data y meaning that for every sequence $\{x_n\} \subset X$ where $Kx_n \to Kx$ as $n \to \infty$ we have $x_n \to x$ as $n \to \infty$,*

and the lack of these properties means the ill-posed nature of the problem.

A model that describes best a parameter-dependent dynamical system is the following

$$\frac{dz}{dt} = g(t, z; \theta), \quad \text{given that} \quad z(t_0) = z_0 \tag{4.2}$$

where

g a known procedure, or mapping,

z_0 the initial state of the system,

$\theta \in \Theta$ the space or set of parameter values in consideration,

$z(t) \in \mathbb{R}^n$ the observations we make which depend on time t.

If we have a direct problem, we have given parameters θ and initial state z_0 so that we determine $z(t)$ for $t \geq t_0$, while in the inverse problem case, we have given solutions $z(t)$ for $t \geq t_0$, so as to find $\theta \in \Theta$.

Now, since we partially know the observations $z(t)$, we can model them by an observation equation, such as

$$f(t, \theta) = \mathcal{H} z(t, \theta), \tag{4.3}$$

with $\mathcal{H}$ being the observation operator, with the best case being the operator coinciding with the identity.

Having a finite number of solutions for our problem, described as

$$\{\tilde{y}_j\}_{j=1}^{p}, \tag{4.4}$$

we can approximately connect them with the previous equation as

$$\tilde{y}_j \approx f(t_j, \theta). \tag{4.5}$$

with the approximate equation including the possible errors due to measurments.

This way we constructed, in an abstract way, the inverse problem based on the known values of our observations and solutions. Numerous examples of such problems are presented in [7].

We can describe a certain common formulation for inverse problems, in a similar way to (4.2), and end up in a more general relation. Supposing the model parameters m and the given data d are vectors, such as

$$m = (m_1, \ldots, m_p) \in \mathcal{M},$$

$$d = (d_1, \ldots, d_n) \in \mathcal{D},$$

with $\mathcal{M}$ and $\mathcal{D}$, the parameter and data space respectively, we introduce a mapping $G : \mathcal{M} \to \mathcal{D}$, which is defined by the equation

$$d = g(m), \tag{4.6}$$

where $g \in G$ is an operator that describes best the model we took the observations from. This way, the approximation in (4.6) can take the form of an integral, a differential equation, or even a linear system.

To finalize the process, we fit the observations $y = (y_1, \ldots, y_r)$, that we obtain by a similar mapping $H : \mathcal{D} \to \mathcal{H}$. Here, $\mathcal{H}$ is the observations space, and, for an observation operator $h \in H$, we have

$$y = h(d) = h(g(m)), \tag{4.7}$$

and, this way, we have effectively created a general relation that describes every inverse problem in a more general notation.

We can divide the general idea of an inverse problem into two separate instances:

- Linear inverse problems, where (4.6) is a linear equation or system, a particularly important case, in which we, often, end up after the discretization of PDEs or integral equations,
- Nonlinear inverse problems, in which case we mostly make use of nonlinear differential and algebraic operators.

Solving the constructed problem is also a challenge when it comes to using the most effective method error-wise. In the first case of linear problems, we can, according to the sparsity and structure of the matrices involved, use either implicit or explicit algorithms, also known by linear algebra and numerical analysis. Examples of these algorithms are Least-Squares solution, L_1-norm solution, as factorization (explicit) methods, and gradient descent, stochastic gradient descent and conjugate gradient methods, as iterative (implicit) methods.

Considering nonlinear problems, since we know only a finite number of measurements, we introduce a residual function that maps the observation vectors of $\mathbb{R}^n$ into $\mathbb{R}$ and use iterative methods in order to optimize the nonlinear system by minimizing this residual function. Similarly to the previous case, examples of these methods are mainly Newton's method, Gauss-Newton, and Levenberg-Marquardt.

A numerical way to approximate the solutions of both linear and nonlinear inverse problems is the Finite Element Method. In both circumstances, we have the constructed system as the input, which, as we thoroughly described in section 2.4, discretizes the differential equation, and creates a linear system that can be easier to solve, along with the visual presentation of these solutions, either they are time-dependent or not, as the output.

4.2 Data Assimilation

In order to implement properly the above-mentioned methods, after effectively creating an inverse problem, we need initial or other conditions. These, as we stated before, can be produced through the process of data assimilation, which is approached in either a variatonal or a probabilistic way. Firstly, in order to give a definition for data assimilation, we refer to [31].

Definition 4.2.1. *Data Assimilation (DA) is the approximation of the true state of some physical system at a given time, by combining time-distributed observations with a dynamic model in an optimal way.*

In both approaches stated above, and the hybrid produced by their combination [33], we aim to find an optimal solution that minimizes a cost function or has a minimum variance, respectively, with the first being less time-consuming and complex, and the latter providing a more detailed information structure. We will focus on the variational aspect of the DA, although sources for the statistical part of the DA can be found in [32].

4.2.1 Notation

To define the variational data assimilation method, we need first to introduce a standard notation that describes best our problem.

Considering a discrete model, that is easier to construct and describe, we suppose the governing state equation

$$x(t_{k+1}) = M_{k+1}\left[x_k\right], \tag{4.8}$$

between two consecutive times t_k and t_{k+1}, where x is the vector that describes the state of the model and M the, in certain instances time-dependent, corresponding dynamics operator, which is the outcome of a finite difference, or the discretization process of a PDE during the implementation of the finite element method.

Our observations and measurements can be modeled in time t_k by

$$y_k^o = H_k\left[x^t(t_k)\right] \tag{4.9}$$

where H is an observation operator and can be, similarly to M, time-dependent. We need to note here that, due to the nature of the inverse problem and the lack of definite knowledge, the dimension p_k of our observation vector $y_k^o = y^o(t_k)$ is significantly smaller than the state dimension n, resulting in $p_k \ll n$.

In both (4.8) and (4.9) we use certain indexes:

"o" as the observation index,

"t" as the unknown true state index,

Introducing now the continuous time-dependent system, we have the equations from before to become

$$\dot{x} = \mathcal{M}(x, t), \tag{4.10}$$

and

$$y^o(t) = \mathcal{H}(x^t, t), \tag{4.11}$$

where we use the $\dot{x}$ as another way to express d/dt, and now the $\mathcal{M}$ and $\mathcal{H}$ are nonlinear operators, respectively, for continuous time and observations, making x and y also functions that are continuous in time.

At this point, we need to note that, when solving PDEs, the state variable takes the form of $u(x,t)$, where x denotes the space variables, to which our system is also dependent, and the space-state operator becomes

$$\mathcal{M} = \mathcal{M}\left[\partial_x^\alpha, u(x,t), x, t\right], \tag{4.12}$$

with ∂_x^α being the partial derivatives regarding the space variables.

4.2.2 Formulation

To formulate a DA variational problem, we begin by supposing a system of PDEs in a region Ω

$$\begin{cases} \dfrac{dx}{dt} = \mathcal{M}(x) & \text{in } \Omega \times [0,T], \\ x(t=0) = x_0, \end{cases} \tag{4.13}$$

and, to find the initial conditions for the system, we will construct an optimal control problem. Since we have the observations $y(t) \in \mathcal{O}$ and an observation operator $\mathcal{H}$ that best describes them, we suppose the cost function

$$J(x_0) = \frac{1}{2}\int_0^T \|y(t) - \mathcal{H}(x(x_0,t))\|_{\mathcal{O}}^2 \, dt + \frac{1}{2}\|x_0 - x^b\|_{\mathcal{X}}^2, \tag{4.14}$$

where $\mathcal{O}$ and $\mathcal{X}$ are the observation and space-time function spaces respectively, x^b is the first guess or else observation and the second part after the integral acting as a regularization.

To summarize the DA variational problem for the optimal analyzed state x_0^a:

Find the x_0^a that minimizes J and satisfies:

$$x_0^a = \arg\min J(x_0), \tag{4.15}$$

to which we provide the necessary condition that also holds

$$\nabla J(x_0^a) = 0. \tag{4.16}$$

To find x_0^a certain algorithms are made use of such as the Adjoint method, 3D-Var and 4D-Var iterative algorithms that make use of gradient descent to update the value of x_j and calculate the optimal value for the cost function.

4.3 Reduced Order Methods

As the complexity of algorithms and computational models increases, there is a growing demand for faster methods of developing and implementing these algorithms. This demand has prompted the development of various ROMs, as we saw in the previous chapter, which seek to reduce both computation time and memory requirements. SVD, POD, PGD, and DMD are some major techniques in ROMs and have been used to speed up computational processes in a variety of applications. The basic idea behind ROMs is that a one-time, computationally intensive setup phase is performed offline, after which the iterative reduction process within the main loop can proceed efficiently, significantly increasing the overall speed of simulations [34].

ROMs typically use a two-loop structure: an inner loop and an outer loop. The inner loop focuses on modeling the core physical problem, which is typically solved using a discretized system of Partial Differential Equations (PDEs). This process frequently involves numerical techniques, which can be resource-intensive. The outer loop, on the other hand, is primarily responsible for calculating optimal parameter values, creating optimization problems, and utilizing the inner loop's results for purposes such as data assimilation or inverse problem solving. The outer loop thus relies on multiple iterations of the inner loop, which can be computationally expensive.

To reduce computational costs, low-fidelity models are frequently used in the inner loop. These models are intended to provide approximate solutions with significantly lower computational requirements [35]. POD, DMD, and machine learning-based approaches, such as Support Vector Machines (SVMs), are popular low-fidelity models because they provide efficient solutions while sacrificing accuracy. High-fidelity models, which provide more accurate representations of the system, are commonly used in the outer loop to maintain the precision needed for optimal decision-making or complex problem-solving. Multi-fidelity systems are formed by combining low-fidelity models for efficient computation in the inner loop and high-fidelity models for accuracy in the outer loop. These systems strike a balance between computational efficiency and model accuracy, resulting in faster and more reliable analysis [36].

In essence, ROMs and multi-fidelity systems help optimize computational resources by combining the strengths of low- and high-fidelity models, increasing the speed and efficiency of solving complex problems in a variety of scientific and engineering applications.

4.4 Machine Learning

Modern-day science benefits massively from the current evolution of machine learning in a way that several branches of it have been developed to provide assistance or even solutions to more difficult scientific problems, and are called Scientific Machine Learning.

Definition 4.4.1. *Scientific machine learning (SciML) is a combination of traditional computer science and engineering approaches with machine learning approaches for the solution of real-world problems. [7]*

This way, by combining model-based and data-driven approaches, we can construct a solid base for the DTs [37].

SciML can be divided into three main categories: Supervised, Unsupervised, and Reinforcement and all of these techniques are used to solve complicated partial differential equations (PDEs).

4.4.1 Supervised Learning

When labeled data is available, supervised approaches can be used to train a model that maps outputs from independent variables [38]. Let $(x_1, y_1), \ldots, (x_n, y_n)$ be a labeled dataset, where x_i is the input vector and y_i its corresponding label, with the main target being the formation of a parametric function f that can map other vectors x to their respective outputs y. This parametric function is presented as

$$f(x; \theta) = y, \tag{4.17}$$

where θ is the set of parameters to be considered for the proper fitting of the model. These parameters θ are chosen in an optimal way so that the occurring prediction error on the training data is minimized. The minimization problem described can be also presented as:

Find the values of $\hat{\theta}$ that minimize the function:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i; \theta))^2, \tag{4.18}$$

and after fitting the model with the optimal values of θ, unidentified information can be predicted.

Supervised SciML has a notable range of applications such as classification, decision making, damage reduction, anomaly detection, etc., with each of them playing a critical role in the construction of a DT, while also reinforcing the core of a DT by helping with procedures like uncertainty quantification, denoising of the data, optimization and more [39], [40].

Numerous advantages of Supervised SciML contribute to its efficacy in a range of applications. When trained on well-labeled datasets, supervised SciML models are known for their exceptional accuracy and consistent predicted performance, while the supervised learning objectives' clarity further helps the evaluation process by making it easier to interpret and evaluate model performance.

On the other hand, Supervised SciML has a number of drawbacks that limit its usefulness in some contexts. Its key disadvantage is that it relies on large, accurately labeled datasets, which can be costly and time-consuming to obtain. Furthermore, these models generally struggle to generalize and are in danger of overfitting or underfitting when new data is added. Also, supervised models are subject to noise, which can degrade their performance, and as they become more complex, they become more difficult to understand, complicating validation and transparency. Finally, the time-consuming and expensive manual annotation process required for huge datasets adds to the implementation burden.

4.4.2 Unsupervised Learning

Unlabeled datasets can be analyzed and clustered using Unsupervised SciML techniques. These algorithms overcome one of the fundamental drawbacks of supervised algorithms, namely the necessity for labeled data, by revealing hidden patterns, or data clusters, without requiring human intervention [41]. To put it another way, an explicit output set does not drive the learning process, and these strategies use the learned features to cluster data, in the form of matrices, in a totally automated fashion [42].

Let $X = \{x_1, \ldots, x_n\}$ be a dataset of n observations, where x_i is a d-dimensional vector of $\mathbb{R}^d$. The target of the method is to create a partition of the above dataset $S = \{S_1, \ldots, S_k\}$, where S_i are k clusters, with k being predetermined. To do this, we need the set of the corresponding cluster centers $C = \{c_1, \ldots, c_k\}$, with c_i being also d-dimensional vectors of $\mathbb{R}^d$. This way, the objective of the unsupervised method, which is the minimization of the distances between observations and their corresponding cluster center, can be described by

$$\min \sum_{i=1}^{k} \sum_{x \in S_i} |x - c_i|^2, \tag{4.19}$$

with $|\cdot|$ being the standard Euclidean distance. The challenge, here, is the identification and detection of said cluster centers that minimize the above problem, which can be accomplished by implementing optimization

techniques like k-means, hierarchical clustering, etc. An example of the use of Unsupervised SciML can be found in [43].

The advantages that Unsupervised SciML provides, due to the lack or requirement of labeled datasets, start with better adaptability and cost-effectiveness, this way making it a more suitable option for different types of models. Moreover, it can process diverse types of data without the constraints of predefined labels, while it is also more effective in terms of anomaly detection, which is crucial for tasks that require analysis.

However, while the lack of labeled data in unsupervised learning improves scalability, it can also limit its usefulness. Without instruction, these algorithms may find patterns unrelated to the goal. Evaluating unsupervised models is difficult since there are no clear indicators, making performance assessment subjective and difficult to understand, as well as the abstract patterns formed can be difficult for people to comprehend, and they are sometimes impacted by data noise, resulting in erroneous patterns. Furthermore, certain algorithms are computationally costly, necessitating substantial resources for huge datasets.

4.4.3 Reinforcement Learning

A rather famous method used in engineering these days, Reinforcement SciML uses trial-and-error methods implementing various dynamic conditions to identify effectively a behavior of the situation encountered, as this is also a common feature of a DT [44]. Reinforcement methods make use of ways to "read" the environment around them as well as the activity there, which are used as the input. The output is the actions that are selected in order to cause an alteration in the current environment's condition in a controlled way. The agent, that makes these changes come through, then computes the sum of the quantities of a reinforcement signal that is used as an indicator of the system's condition. This way, through continuous implementations, the agent learns to act efficiently over time [45], [46].

To do this, considering the state of our system to be $s_t \in S$ in a given time point t, where S is the set of possible states, and the actions $a_t \in A$ from which the agent selects any one to implement, where A is the set of possible actions, we introduce a numerical reward r_{t+1}, that is given to the agent based on the action that it picks to send the system from s_t to s_{t+1}. We aim in the construction of an optimal policy $\pi(a_t, s_t)$ that consists of actions of the agent, that maximize the overall reward. Thus, we, first, create a value function that we maximize, in order to get the aforementioned optimal

reward, and we form the following:

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \Big| s_0 = s, \pi\right], \tag{4.20}$$

for a random policy $\pi(a_t, s_t)$, where γ is a factor that indicates the importance of the reward at each time step to the cumulative reward.

We make use of (4.20) in order to form the action-value function, which measures the reward we have by starting from the state s after implementing the action a following the optimal policy

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \Big| s_0 = s, a_0 = a, \pi\right], \tag{4.21}$$

where this optimal policy is obtained by

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_{a' \in A} Q^*(s,' a), \\ 0, & \text{otherwise.} \end{cases} \tag{4.22}$$

Taken together, (4.20) and (4.21) form the Bellman equations

$$V^*(s) = \max_{a} Q^*(s, a), \tag{4.23}$$

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r_{t+1} + \gamma \max_{a'} Q^*(s', a')\right], \tag{4.24}$$

where s' and a' are the next state and action, and Reinforcement SciML methods use these equations in an iterative way in order to find the optimal value and action-value functions and maximize the output.

Reinforcement SciML provides us with several advantages such as, without so much as explicit programming, the enabling of agents to judge in an autonomous way, which is precious in cases of complex problems where setting rules is impossible, the adaptability of said agents which helps in configuring new strategies and implement them on different kinds of problems. Real-world situations are also benefitted since they use mostly high-dimensional state and action spaces, while the sequential decision-making character of the method makes it perfect for problems that span multiple phases.

On the other hand, there are, also, quite some drawbacks to reinforcement learning. Initially, reinforcement learning is frequently sample-inefficient, needing multiple interactions with the environment to acquire efficient rules. Another difficult part is striking a balance between exploitation (choosing

known effective actions) and exploration (trying random actions). Also, in order to achieve optimal rewards, safety, and ethics are sometimes ignored, while trying to create a more general knowledge that surrounds several tasks and environments too.

4.5 Complex Structures within Digital Twins

The advantages that emerge from the use of SciML are truly substantial. SciML can be applied to each and every one of the aforementioned steps for the creation of a DT, and, this way, we can create complex structures that comprise methods that solve PDEs, and SciML techniques that provide the former methods with supportive data, optimize the procedure through the implementation of ROMs, or, lastly, when we lack knowledge about the governing PDEs, machine learning can bridge the gap between the two.

PDEs are required to create digital twins because they provide a mathematical framework for characterizing the behavior of physical systems. Solving PDEs allows engineers and scientists to create mathematical models, that accurately capture the physical behavior of complex systems. This allows them to forecast how the system will behave and perform under various scenarios. In general, PDEs in digital twins cannot be solved quickly enough using conventional methods.

However, the application of SciML has recently generated a lot of interest, particularly in light of the digital twin concept. This is because PDEs, which can be extremely complex and computationally demanding, can be solved accurately and efficiently using machine learning techniques. Scientific machine learning is an interdisciplinary field that draws on concepts from scientific computing, machine learning, and numerical techniques. It aims to create machine learning algorithms tailored to scientific problems in biology, chemistry, physics, and other fields. These algorithms address scientific computing challenges such as precision, resilience, and interpretability.

The use of neural networks has yielded impressive results in this field in recent years. In this context, there are three main methods: Physics-Informed Neural Networks, Deep Energy Methods, and Neural Operators [47].

4.5.1 Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) are a machine learning methodology that can be used to solve partial differential equations. The fundamental concept of PINNs is to train a neural network to approximate PDE

solutions, by minimizing a loss function that considers the PDE residual at particular locations within the domain (collocation points), as well as the initial and boundary conditions over the boundary of the space-time domain. PINNs are deep-learning neural networks that, after training, generate an estimated solution to the differential equation, given an input point in the integration domain. The addition of a residual network that encodes the governing physics equations is a noteworthy innovation of PINNs. Interestingly, PINN training does not use labeled data, such as outcomes from previous experiments or simulations, and instead uses an unsupervised approach. The main characteristic of PINN is that it solves PDEs without the use of meshes by transforming the governing equations' solution into an optimization problem for the loss function [48].

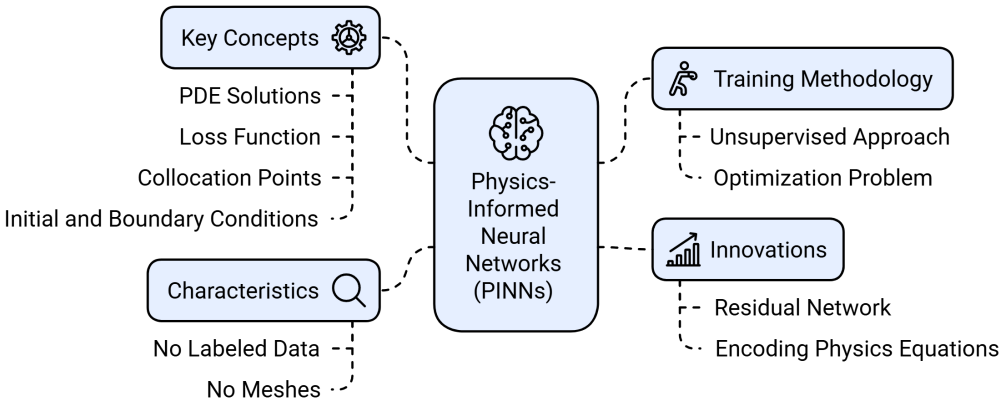


Figure 4.1: The Physics-Informed Neural Networks concept

To successfully describe the methodology, we consider our PDE to be of the form:

$$\mathcal{L}(u(x,t)) = 0, \tag{4.25}$$

where $\mathcal{L}$ is a differential operator, $u(x,t)$ is the solution of the PDE, and x and t are vectors that represent space and time respectively.

In order for the PINN to operate and solve the PDE above, we define the neural network, that will approximate our solution $u(x,t)$, by

$$f_\theta(x,t), \tag{4.26}$$

which also operates on space and time, with θ being the parameter vector for the proper setting of the neural network.

The loss function, that the PINN is designed to minimize, is defined as

$$\mathcal{L}_{PINN}(\theta) = \lambda_1 \mathcal{L}_{data}(\theta) + \lambda_2 \mathcal{L}_{phys}(\theta), \tag{4.27}$$

in which the PDE is integrated. Here, we have λ_1 and λ_2, which are the corresponding weights for $\mathcal{L}_{data}(\theta)$ and $\mathcal{L}_{phys}(\theta)$, that, in turn, are loss functions that calculate the divergence, as the mean squared error (MSE), between the neural network results and the available data, or the actual solution of the PDE, respectively. The first is described as

$$\mathcal{L}_{data}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(u(x_i, t_i) - f_\theta(x_i, t_i))^2, \tag{4.28}$$

considering the available data points $(x_i, t_i, u(x_i, t_i))$, that are N in number. Accordingly, the latter, given the M number of collocation points, is defined by

$$\mathcal{L}_{phys}(\theta) = \frac{1}{M}\sum_{j=1}^{N}(\mathcal{L}(f_\theta(x_j, t_j)))^2, \tag{4.29}$$

in which the PDE is incorporated.

At this point, in order to minimize the loss function (4.27), we make use of optimization methods that rely on gradients, with respect to different values of the parameters θ, only to result in an approximation $f_\theta(x, t)$ that combines knowledge from both the PDE and the available data, which it also satisfies.

Real-life examples, that make use of PINNs, are DTs for nuclear reactors [49], air pre-heaters [50], or even medical devices [51].

PINNs are very data-efficient and can operate on smaller datasets, making them useful in situations where data collecting is expensive or impossible, while their adaptability enables them to manage complex, nonlinear interactions, making them ideal for applications requiring detailed physics. However, there are certain downsides to PINNs, particularly the high processing costs associated with solving partial differential equations (PDEs) during training, as well as the requirement of domain knowledge to include precise physics and restrictions, and being susceptible to hyperparameter adjustment. To put it in one phrase, PINNs are powerful, they must be used with caution to balance their strengths and limitations.

4.5.2 Deep Energy Methods

An alternative to data-driven inference for physical systems employing PINNs is the Deep Energy Method (DEM). In contrast to PINNs, which employ PDEs and reduce the corresponding residual, this method minimizes the system's potential energy [52]. This method was swiftly embraced and used in many other sectors, despite the fact that it is only relevant to physical

systems that follow the minimum potential energy principle. The process is simple: first, the energy must be defined. Next, a conventional optimizer is used to minimize the loss function, also known as the energy functional, and determine the deep neural network's biases and weights. In order to arrive at the final prediction, the method first uses forward propagation to approximate the unconstrained solution before applying boundary conditions to constrain it. The system's potential energy is regarded as the loss function, and the numerical integration techniques employed determine how accurate the solution is. The trained network forecasts the outcomes for PDEs at random locations across the domain when the network parameters are at their ideal levels [53].

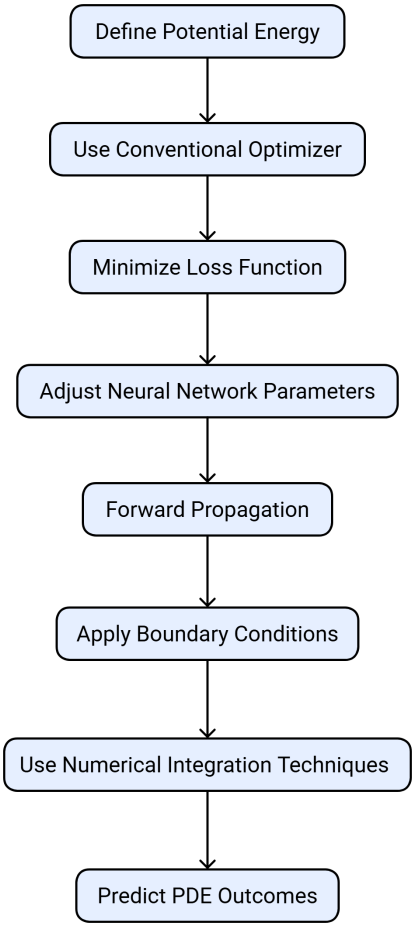


Figure 4.2: Construction of a Deep Energy Method model.

Same as before, we need to first define a steady-state PDE:

$$\mathcal{L}(u(x)) = f(x), \quad x \in \Omega, \tag{4.30}$$

with, again, $\mathcal{L}$ being the differential operator. In this case, we have $u(x)$ as the unknown function and solution of the PDE, $f(x)$ as a given function, and Ω the domain that we are interested in.

In order to compute the approximate function of the solution, we make use of a deep neural network, and notate the approximation as $u_\theta(x)$ using θ as the parameters of the network, like weights and biases. Here, we use as input only the coordinates of space, and the output is the desired approximation of $u(x)$.

The loss function, which we want to minimize, in this instance, is defined by:

$$E(\theta) = \int_\Omega \Psi(\epsilon(u_\theta))\ d\Omega, \tag{4.31}$$

with $\Psi(\epsilon(u_\theta))$ being the system's potential energy, as assessed at the domain Ω. We end up with the optimization problem, which is stated as

Find the value of the parameters θ^* in:

$$\theta^* = \arg\min_\theta E(\theta), \tag{4.32}$$

locating, this way, the optimal values of weights and biases for the training of the deep neural network.

Examples, where Deep Energy Methods apply in real-life phenomena, are Hamiltonian mechanics [54], the Landau theory [55], and phase-field modeling [56].

Unlike purely data-driven approaches, DEM leverages physical principles during training to integrate neural networks with physics-based modeling, same as PINNs, to produce predictions that are more accurate. Because DEM only requires the first derivatives, it is more effective than PINNs at solving second-order PDEs. It is perfect for applications involving complex physics because it is data-efficient and adapts effectively to complex, nonlinear interactions. Because DEM effectively improves network parameters through loss minimization, it is particularly useful for engineering issues requiring energy functionals.

The first of DEM's significant drawbacks is integration errors, which can lower forecast accuracy, especially when it comes to collecting displacement and stress concentrations, which are important in engineering applications. Additionally, not all governing equations fall into an energy-minimization

framework, which limits its usefulness. Furthermore, the technique for numerical integration within the loss function determines how accurate the DEM is, and careful calibration of hyperparameter sensitivity is necessary for best results.

4.5.3 Neural Operators

The traditional approach to neural network development has focused mostly on obtaining knowledge about mappings between finite-dimensional Euclidean spaces or finite sets. To achieve more precise approximations of the solution operators that arise in the setting of PDEs, a more extensive type of neural network has been proposed to learn operators that map between infinite-dimensional function spaces. These operators are approximated by combining a collection of layers and nonlinear functions, allowing the model to accurately mimic complicated nonlinear operators [57].

In other words, the Neural Operator is a cutting-edge deep learning architecture meant to learn about an operator that acts as a mapping across infinite-dimensional function spaces. This unique methodology can be used to solve PDEs by training a neural network to learn an operator that transfers an infinite-dimensional space (u, t) to another infinite-dimensional space $f(u, t)$, instead of resolving the PDE problem using the finite element method. The neural operator learns a continuous function that maps between two continuous function spaces, allowing for more precise solutions to PDE problems [58].

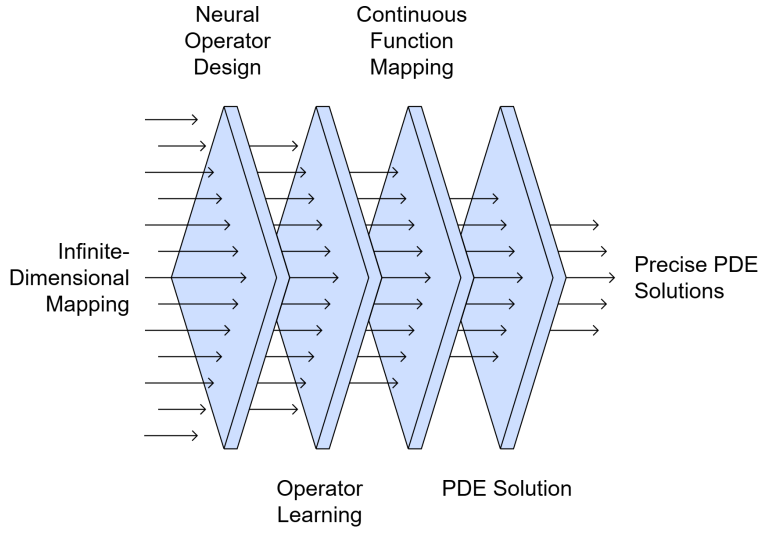


Figure 4.3: Construction of a Neural Operator.

For this category of methodologies, we consider our system of parameterized PDEs to have the form of:

$$\mathcal{L}(u, s) = 0, \tag{4.33}$$

with u and s being the input parameters and the output solutions to the PDEs, respectively. We make the assumption that every input function u is linked to a corresponding solution s to the PDEs, considering the relevant initial conditions. Moreover, we define a solution operator by

$$G(u) = s(u), \tag{4.34}$$

which will be later used by each class of neural operators to obtain the optimized solution.

Several types of Neural Operators (NO) are known to have been developed such as:

Fourier NO

Fourier neural operator [59] learns a neural operator with a Dirichlet kernel. It is based on Fourier transforms across infinite-dimensional function spaces and learns better than the other types in some cases. This operator restricts the integral operation to a convolution process, effectively realized through a linear transformation within the Fourier domain, and is better used on PDEs like the Navier-Stokes equation outperforming several other deep learning methods.

Markov NO

Markov neural operators [60] use Markov processes to learn a neural operator. By connecting the operators into a Markov chain, it can forecast the subsequent steps with little loss using only single-step learning information and can be useful in situations where the instability of a system creates chaos and unpredictability.

Laplace NO

Training a neural network [61] to learn the Laplace operator, a basic mathematical operator that shows up in many significant differential equations, is known as the Laplace neural operator. By forecasting the values of the Laplace operator at different places in the domain, the neural network can be used to estimate the solution of a differential equation after it has been trained. This technique is proven to be more successful in cases of parameterized PDEs using Laplace operator

eigenfunctions on Euclidean spaces, although can be unreliable in real-time problems due to the complex domains existing.

Deep Operator Network (DeepONet)

Two subneural net structures known as the branch network and the trunk network assist the DeepONet [62] in learning a neural operator. While the trunk receives the sites where the output is assessed, the branch network receives the input data at predetermined sensor points.

Here, the solution operator, stated in (4.34), is noted as G_θ due to the existence of the parameters θ that will be used to train the NO. In general, in DeepONet instances, we define the branch network as a procedure that uses the input u in the form of a vector of function values $u = [u(x_1), \ldots, u(x_m)]$, with $\{x_i\}_{i=1}^m$ a specific collection of locations within the domain, and gives the output vector $[b_1, \ldots, b_q]^T \in \mathbb{R}^q$ that describes a set of features existing in the system. The trunk network, on the other hand, uses as input y, which represents continuous coordinates within the domain, and as an output, it generates a vector $[t_1, \ldots, t_q]^T \mathbb{R}^q$, that also describes features of the system. Hence, G_θ takes the form of:

$$G_\theta(u)(y) = \sum_{k=1}^{q} b_k(u(x_1), \ldots, u(x_m))\ t_y(y), \tag{4.35}$$

with the first part representing the branch, and the second the trunk procedure respectively.

To create a physics-informed DeepONet, we use a similar to the cost function of (4.27):

$$\mathcal{L}(\theta) = \lambda_1 \mathcal{L}_{operator}(\theta) + \lambda_2 \mathcal{L}_{phys}(\theta), \tag{4.36}$$

and infuse the above equation, hence getting the corresponding relations:

$$\mathcal{L}_{operator}(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| G_\theta(u^{(i)})(y_{u,j}^{(j)}) - G(u^{(i)})(y_{u,j}^{(j)}) \right|^2, \tag{4.37}$$

$$\mathcal{L}_{phys}(\theta) = \frac{1}{NQm} \sum_{i=1}^{N} \sum_{j=1}^{Q} \sum_{k=1}^{m} \left| \mathcal{L}(u^{(i)}(x_k), G_\theta(u^{(i)})(y_{u,j}^{(j)})) \right|^2. \tag{4.38}$$

Above, we have $\{u^{(i)}\}_{i=1}^{N}$ as the distinct input function values that are N in number, and $\{y^{(j)}\}_{j=1}^{P}$ as the locations within the domain of the solution $G(u^{(i)})$, that we get from observing the data, such as initial or boundary conditions, and are P in number. This way, (4.37) takes the algorithm close to the desired data of the solution, and at the same time (4.38) makes sure the constraints, that are needed for the PDE to be properly described, are met.

Applications of this method are met in several fluid instances, due to the high complexity of underlying physics, or in cases such as weather forecasting, CO2 emission migrations, etc.

Wavelet Neural Operator (WNO)

This approach, [63], blends wavelet transformation and integral kernel methods. Real-time prediction of complex systems is made possible by its direct learning of the nonlinear mapping between functional spaces. WNO precisely captures minute features and relationships in complicated issue solutions by utilizing the wavelets' high spatial and frequency resolution and capacity to track patterns in the spatial domain.

Neural Operator approaches have particular advantages, making them useful for a variety of computational applications. They are computationally efficient, able to solve specific PDEs rapidly and consistently across spatial resolutions due to their resolution invariance. Their discretization invariance improves adaptability to various numerical schemes, and they excel at zero-shot super-resolution, which allows for precise extrapolation in untrained regions. Neural operators also approximate mappings between infinite-dimensional Banach spaces and can handle high-frequency input, making them well-suited to dynamic phenomena.

However, obstacles include interpretability issues due to black-box parameterization, the possibility of vanishing gradients during training, and potential generalization errors that can impair accuracy on out-of-sample data.

Concluding, whether to use PINNs, DEMs, or NO is determined by criteria such as data availability, computational resources, physics inclusion requirements, and tolerance for interpretability or generalization difficulties. Each method's strengths correspond to different problem requirements.

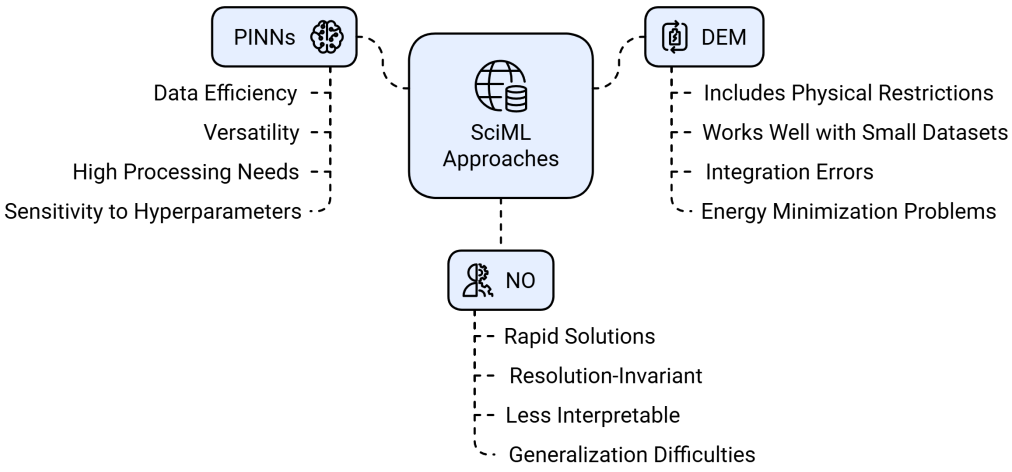


Figure 4.4: All instances of Scientific Machine Learning.

4.5.4 Modern-Day Implementations

PINNs and Poisson Equation

In [64], several types of PINN constructions are introduced and implemented in different examples. One of them is the solution of the Poisson Equation using a neural network to predict an approximation and a loss function that is obtained by imposing constraints on the boundary of the domain $\partial\Omega$.

The neural network that we use is a Multi-Layer Perceptron (MLP), which is fairly common, in order to obtain optimal values for the parameters θ of the non-linear approximation function u_θ of the target solution $u(x)$. This neural network is organized into a sequence of $L+1$ layers, with the first $\mathcal{N}^0$ being the input layer, and defined by:

$$\mathcal{N}^0(x) = x, \tag{4.39}$$

with x as the input vector. After this, for each subsequent layer l, there is a weight matrix W^l, with dimensions $d_{l-1} \times d_l$ and a bias vector $b^l \in \mathbb{R}^{d_l}$, where d_l is the output size of layer l. All the layers l, from $l = 1$ up to $l = L - 1$, are the hidden layers, and the output for each of them, using a non-linear function σ, or activation function, is defined by:

$$\mathcal{N}^l(x) = \sigma(W^l \mathcal{N}^{l-1}(x) + b^l), \tag{4.40}$$

and the final layer of the neural network, or the output layer, is defined by:

$$\mathcal{N}^L(x) = W^L \mathcal{N}^{L-1}(x) + b^L. \tag{4.41}$$

Lastly, the outcome of the MLP, u_θ, is defined by $u_\theta(x) = \mathcal{N}^L$, while we can, also, describe it using the function composition $\circ$:

$$u_\theta = (\mathcal{N}^L \circ \mathcal{N}^{L-1} \circ \cdots \circ \mathcal{N}^0)(x), \tag{4.42}$$

with, finally, the parameters being $\theta = \{W^l, b^l\}_{l=1}^L$.

For this problem, a Vanilla-PINN is defined with the input layer having two input variables/neurons for the two corresponding space coordinate variables x and y, the output layer having a single variable, or neuron, that represents the projected answer,$u_\theta(x, y)$, and, between them, three hidden layers, each of which contains five neurons, for the learning of complex patterns and representations by the neural network. Also, in the last part, it evaluates the partial derivatives required to create the PDE loss function $\mathcal{L}_{PDE}(j)$ through the process of Automatic Differentiation.

This structure can be visualized as follows:

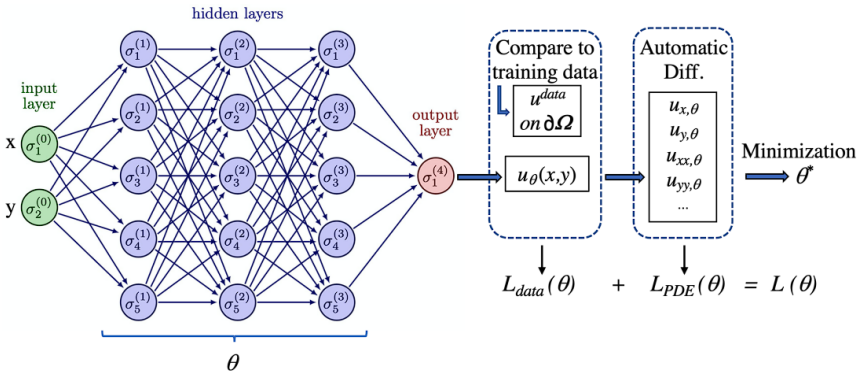


Figure 4.5: Vanilla-PINN for direct 2D PDE - Dirichlet boundary conditions [64].

The number of the parameters that the PINN learns is given by the formula

$$N = i \times h_1 + h_1 \times h_2 + h_2 \times h_3 + h_3 \times o + h_1 + h_2 + h_3 + o, \qquad (4.43)$$

where h_j is the respective number of neurons in the hidden layer j, i is the number of initial neurons, and o is the number of neurons in the output layer, so the parameters learned in this case, and that, ultimately, comprise θ, are 81.

The domain Ω is a square with sides of 1 cm as its boundary $\partial\Omega$. In order for the (4.28) to apply, the generation of a data set is required, for which points located on the four sides/boundaries of the domain are used, and, in particular, 120 points, 30 randomly distributed from each side. Furthermore, there is a need for the generation of a set of collocation points, where 400 points are randomly picked from the inside of the domain.

Specific details about the SciML parameters, that were chosen to optimally solve the problem, are given in [64]. The PDE loss function, stated in (4.29), is based on $\mathcal{F} = u_{xx} + u_{yy} - f(x, y)$, similar to the Poisson equation itself, and the results produced are shown below.

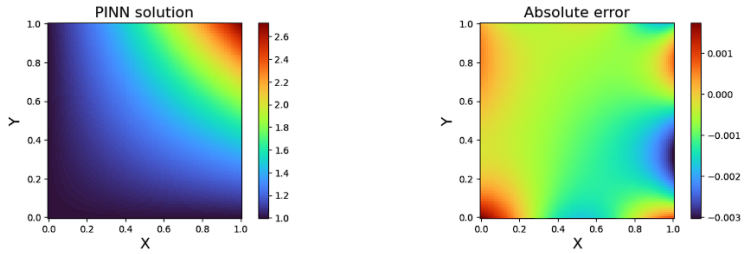


Figure 4.6: Implementation of Vanilla-PINN described in Figure 4.5 (left), absolute error produced (right) [64].

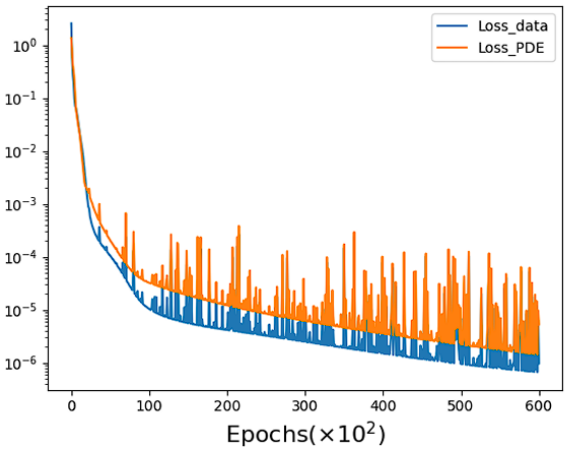


Figure 4.7: Evolution of the loss functions L_{data} and L_{PDE} considering the number of iterations (epochs) [64].

DEM and Thermoelasticity

In [66], a Deep Energy Method is constructed in order to solve the linear static thermoelasticity PDE problem with boundary conditions over parameters such as stress, displacement, temperature difference, considering thermal conductivity, prescribed displacement and temperature, traction, and heat flux.

Suppose the domain is Ω along with the boundary $\partial\Omega$, we first define the Cauchy stress as σ, the displacement as u, $\phi = T - T_{ref}$ being the temperature difference with T_{ref} the reference temperature, and b the body force. We will also use ∇ as the gradient operator along with k as the thermal conductivity and n as the outer norm vector. Lastly, g will be the prescribed displacement on the boundary $\partial\Omega_g$, $\bar{\phi}$ will be the prescribed temperature on the boundary $\partial\Omega_T$, t will represent the traction on $\partial\Omega_t$, and, finally, q the prescribed heat flux on $\partial\Omega_q$. This way, we introduce the governing equations of the phenomenon to be

$$\nabla \cdot \sigma(x) + b(x) = 0, \qquad \forall x \in \Omega, \tag{4.44}$$

$$\nabla \cdot k\nabla\phi(x) = 0, \qquad \forall x \in \Omega, \tag{4.45}$$

$$u(x) = g, \qquad \forall x \in \partial\Omega_g, \tag{4.46}$$

$$\phi(x) = \bar{\phi}, \qquad \forall x \in \partial\Omega_T, \tag{4.47}$$

$$\sigma(x) \cdot n = t, \qquad \forall x \in \partial\Omega_t, \tag{4.48}$$

$$-k\nabla\phi(x) \times n = \bar{q}, \qquad \forall x \in \partial\Omega_q, \tag{4.49}$$

and for the aforementioned boundaries, the relations that hold are the fol-

lowing

$$\overline{\partial\Omega_g \cup \partial\Omega_t} = \partial\Omega, \quad \partial\Omega_g \cap \partial\Omega_t = \varnothing, \quad \overline{\partial\Omega_q \cup \partial\Omega_T} = \partial\Omega, \quad \partial\Omega_q \cap \partial\Omega_T = \varnothing.$$

Another way to express the linear consitutive law is:

$$\sigma = C\epsilon - \beta\phi I, \tag{4.50}$$

where I is the identity matrix, C is the elastic material coefficient, and, lastly, ϵ and β are the strain and the thermolelastic coefficient, respectively.

Similarly to the previous example, an MLP neural network is used and the result is computed as in (4.42), and we have that

$$\hat{z}^l := \sigma^l(W^l \cdot \hat{x}^{l-1} + b^l), \tag{4.51}$$

as the relation between inputs and outputs of the layer l.

The loss function here is defined as

$$\mathcal{L} = U - W, \tag{4.52}$$

where

$$U = \int_\Omega \psi(\hat{\epsilon})\ d\Omega, \tag{4.53}$$

$$W = \int_\Omega fb \cdot \hat{u}(x;\theta)\ d\Omega + \int_{\partial\Omega_t} \bar{t} \cdot \hat{u}(x;\theta)\ \partial\Omega_t. \tag{4.54}$$

with U and W being the strain energy and the external work done respectively.

In order to solve this multi-physics problem, two types of algorithms are constructed, the sequential, where it computes displacement and temperature individually while altering the whole neural network for each phase of the procedure, and the coupled, where displacement and temperature are taken simultaneously for the unification of the phenomenon.

For the 2D implementation of these algorithms, a cross-section of a cylinder is considered, the area between a big circle of radius $r_2 = 2m$, and a smaller one inside of radius $r_1 = 1m$. We, now, suppose that the temperature on the inner wall of the cylinder is $T_{in} = 100°C$, and on the outer wall is $T_{out} = 0°C$, while the material of which the cylinder is constructed has, for its parameters, the values: $E = 210\ GPa$, $\nu = 0.3$, $k_x = k_y = 1\ W/(m°C)$, and $\alpha = 1.2 \times 10^{-5}\ 1/°C$.

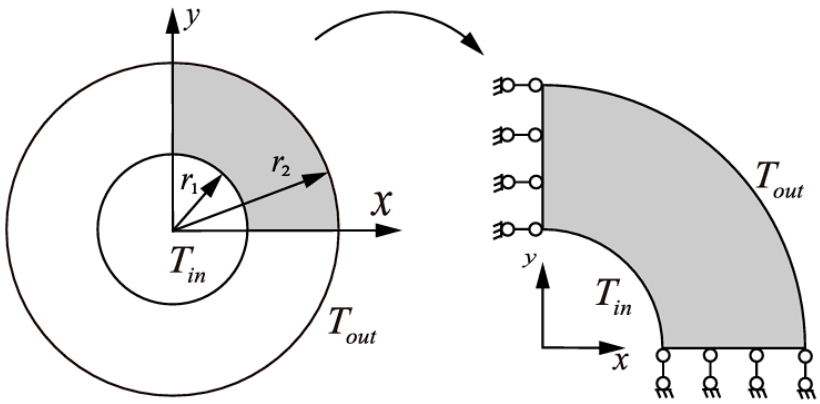
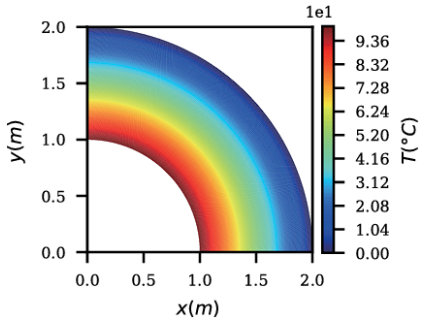
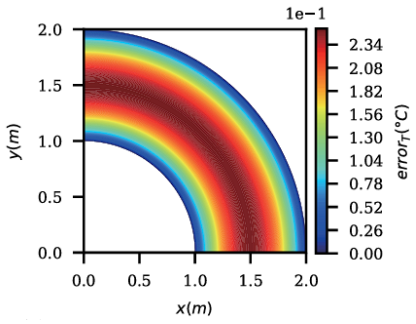


Figure 4.8: Part of the cylinder for the implementation of the thermoelasticity solution [66].

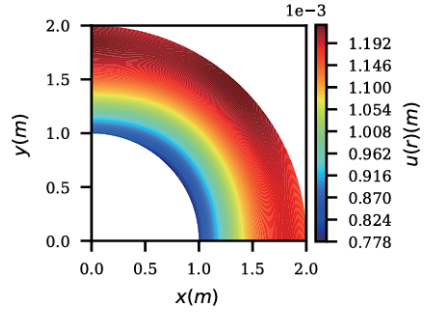
Detailed information about the Dirichlet boundary equations that constrain the neural network, can be found along with the code that is used, on [68] and [69], and the produced results for the constructed Deep Energy Method are presented below.



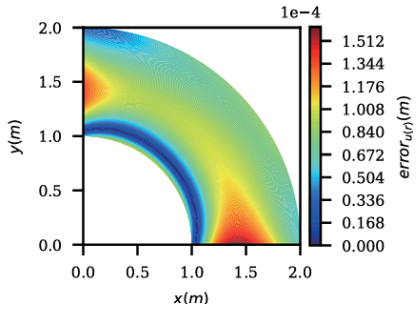
(a) Temperature T.



(b) Error of temperature.

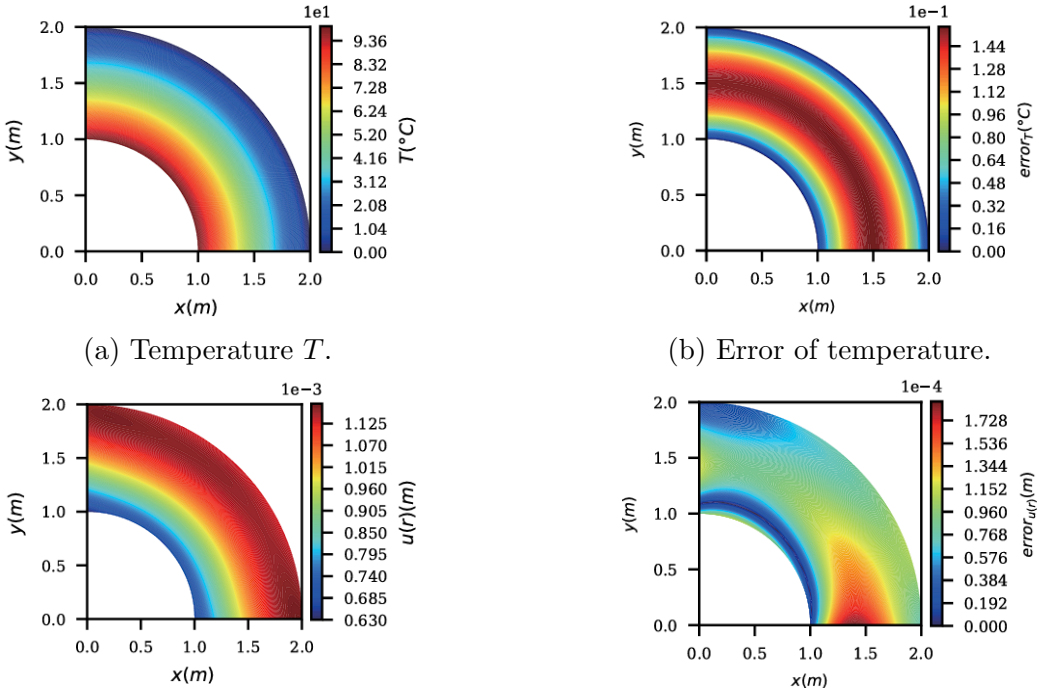


(c) Displacement $u(r)$.



(d) Error of displacement.

Figure 4.9: Thermoelasticity results using the sequential solver [66].



(a) Temperature T.

(b) Error of temperature.

(c) Displacement $u(r)$.

(d) Error of displacement.

Figure 4.10: Thermoelasticity results using the coupled solver [66].

Solver	Variable	L^2 error norm	Training time (s)
Sequential	T	$7.02E-08$	259.54
	u	$7.58E-02$	
	T	$7.02E-08$	247.08
	u	$1.28E-01$	
Coupled	T	$3.26E-04$	400.82
	u	$1.33E-01$	
	T	$4.95E-04$	297.95
	u	$9.04E-02$	

Table 4.1: Sequential and Coupled solver results for different activation functions [66].

DeepONet and Diffusion-Reaction Systems

In [67], the application of physics-informed DeepONets in the solution of various PDE problems is introduced, such as the nonlinear diffusion-reaction PDE, as seen in 2.4.6, although this time with a specific source term $u(x)$,

which is defined as

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad (x,t) \in (0,1] \times (0,1]\,. \tag{4.55}$$

In this instance, the initial and boundary conditions are zero, while $D = 0.01$ and $k = 0.01$ are the diffusion coefficient and the reaction rate respectively.

The neural network here has an architecture, G_θ, that consists of the branch and the trunk network, which are similar and are comprised of 5 layers of 50 neurons for each hidden layer. The idea is to construct an operator that can map different source terms like $u(x)$ to the respective solution $s(x)$ of the PDE. The PDE residual, using an input function $u^{(i)}$, is defined by

$$R_\theta^{(i)}(x,t) = \frac{dG_\theta(u^{(i)})(x,t)}{dt} - D\frac{d^2G_\theta(u^{(i)})(x,t)}{dx^2} - k\left[G_\theta(u^{(i)})(x,t)\right]^2, \tag{4.56}$$

with $\{u^{(i)}\}_{i=1}^N = \{u^{(i)}(x_1), \ldots, u^{(i)}(x_m)\}_{i=1}^N$ representing the input functions, and $\{x_i\}_{i=1}^m$ being a collection of sensors located in different spots in $[0,1]$.

Using the loss function of (4.36), we train the parameters of the neural network minimizing the function

$$\mathcal{L}(\theta) = \frac{1}{NP}\sum_{i=1}^{N}\sum_{j=1}^{P}\left|G_\theta(u^{(i)})(x_{u,j}^{(i)}, t_{u,j}^{(i)})\right|^2 + \frac{1}{NQ}\sum_{i=1}^{N}\sum_{j=1}^{Q}\left|R_\theta^{(i)}(x_{r,j}^{(i)}, t_{r,j}^{(i)}) - u^{(i)}(x_{r,j}^{(i)})\right|^2. \tag{4.57}$$

For the above relation, we uniformly sample, P in number, points, $\{(x_{u,j}^{(i)}, t_{u,j}^{(i)})\}_{j=1}^P$, that lie in the boundary $[0,1] \times [0,1)$, as well as $\{(x_{r,j}^{(i)}, t_{r,j}^{(i)})\}_{j=1}^Q$ that are Q in number, which are the collocation points that satisfy $x_{r,j}^{(i)} = x_j$, and $\{t_{r,j}^{(i)}\}_{j=1}^Q$ lie in $[0,1]$, for each $u^{(i)}$.

The code that makes use of all this methodology, takes also advantage of the conjugate gradient and stochastic gradient descent methods in order to minimize the loss function, with the predictions, as well as the exact and error results being shown below.

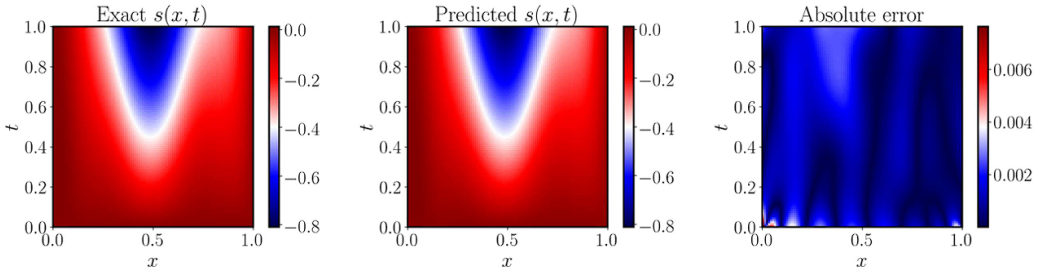


Figure 4.11: Solution of the parametric diffusion-reaction system [67].

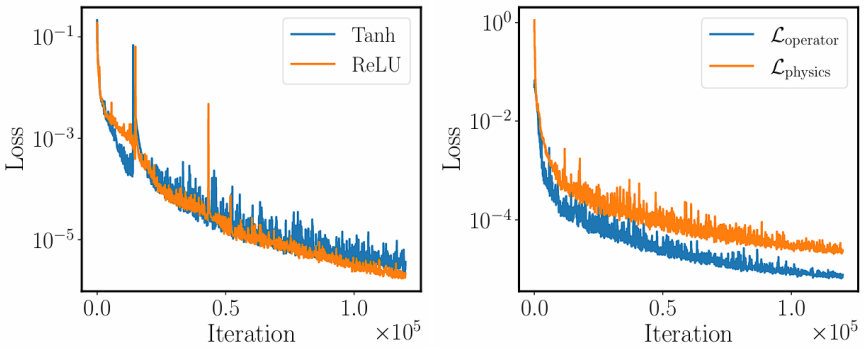


Figure 4.12: Training loss convergence of a DeepONet equipped with different activation functions (left), and a physics-informed DeepONet equipped with Tanh activation function (right) for 120,000 iterations of gradient descent using an Adam optimizer. [67].

Case	Variables	Relative L^2 error	Training time (s)
DeepONet	$s(x)$,$u(t)$	$1.92\% \pm 1.12\%$	4068

Table 4.2: Relative error production and computational cost for training the DeepONet model [67].

4.5.5 Comparing Methods

All in all, we examined two SciML approaches: Physics-Informed Neural Networks (PINNs), and Neural Operator (NO), along with the Finite Element Method imbued with gradient methods, evaluating their distinct benefits and drawbacks, the errors produced and, finally, the time needed for each and every one to come with a result.

FEM is a solid numerical way to solve PDEs in cases where there is a complete set of the differential equation and the corresponding data. If we insert numerical ways that can help reduce the error of the approximate result, it can be one of the most efficient ways to effectively solve and visualize PDEs, systems, or even more complex forms of equations. Where they fall behind is in cases of data unavailability, when we have to intervene with scientific machine learning.

PINNs excel in incorporating physical rules into learning, resulting in data efficiency and versatility, particularly when dealing with inverse situations. However, they have large processing needs due to PDE solving during training, demand extensive physics expertise, and are sensitive to hyperparameters.

DEM effectively includes physical restrictions, and it performs well with tiny datasets and complex data linkages, and it is especially useful for second-order PDEs because it requires only first derivatives. Nonetheless, DEM may induce integration errors and is confined to problems that may be phrased in terms of energy minimization. Like PINNs, it necessitates precise hyper-parameter adjustment.

NO provides rapid, resolution-invariant solutions and can handle a variety of PDEs, making it computationally economical across spatial resolutions and discretization schemes. Its zero-shot super-resolution capabilities allow for predictions beyond trained zones. However, NO models are less interpretable and may exhibit generalization difficulties outside of the training distribution.

As we can see from the implementation of the methods, SciML is better used in cases of complex problems where FEM cannot be really of use. The time needed for the simplest application of a method is disproportionately more, than the difference between the corresponding produced errors.

Chapter 5

Conclusions and Future Goals

FEM provides detailed and precise predictions for a wide range of engineering problems, such as structural, thermal, fluid, electromagnetic, and dynamic simulations, making it an indispensable tool in modern engineering, design, and manufacturing. Instead of solving these problems analytically (which can be extremely difficult or impossible for complex geometries), FEM breaks the problem into smaller, simpler pieces — the finite elements. Each element is solved mathematically, and their results are assembled to form an approximate solution to the overall problem, as presented in Chapter 2. Its key features, the discretization of complex systems, the flexibility in geometry and material modeling, the ability to model multiple physical phenomena, while it also provides us with both global and local analysis to observe either the entire system or critical regions of it, offer significant advantages to the method with its greatest merits being its precision, versatility and real-world applicability.

FEM is an indispensable tool in modern engineering and scientific research, although, in cases where there is high computational complexity, in order to apply the above method properly, RBMs come into action. As we described in Chapter 3, RBM is a model reduction technique used to accelerate the solution of parameter-dependent problems. The key idea behind RBM is that instead of solving a full-scale high-dimensional system (which can be computationally expensive), you create a smaller, reduced model that captures the essential physics of the system but with much less computational effort. Methodologies like POD, PGD, and DMD provide different aspects of the way that the reduced model is produced, and are more effective in diverse types of problems. The offline-online decomposition, the dimensionality reduction, the interplay with FEM, are key features that provide us with more efficient parametric simulations, optimization in terms of shape and topology, and modeling nonlinearities.

Finally, in order to incorporate both of the latter methods in a larger and more complex but efficient structure, in Chapter 4 we introduce DTs, an advanced, real-time representation of physical systems or objects, enabling continuous monitoring, simulation, and optimization of their behavior. By combining PDE solution methods, RBMs, and modern-day SciML methods, they can offer valuable insights into system performance, allowing for predictive maintenance, optimization, and scenario testing, while also enabling customization and personalization in various applications, from product design to patient care. Complex constructs, such as PINNs, DEMs, or NOs, can adapt to different types of problems, according to the proportion of the available data and the complexity of the governing physics, or the general nature of the mathematical interpretation.

Lastly, we conclude by mentioning that the future of Digital Twins lies in the construction of innovations. Fields that can massively benefit from making use of DTs are, mainly, personalized medicine, as a Digital Twin, given its complex structure, key advantages, and adaptability can successfully recreate a cell, a fiber, an organ, or even an entire organism, such as a human, creating hope for the creation of treatments or construction of real-sized, functional organs and fiber, tailored to each individual. Other initiatives of large organizations that are based on DTs are surveillance and monitoring of the environment of our planet. Several models are created and are soon to be created that will eventually result in the creation of a Digital Twin that can replicate the entire Earth along with each and every one of the processes that take place on it.

Acknowledgments

At this point, I would like to express my gratitude to all the people who have been there for me for the duration of the specific Master's program.

First of all, I would like to thank my supervising Professor, Efthymios Karatzas, for providing me with all the necessary stimuli and knowledge, that I needed in order to get where I stand today. I would also like to thank him for the helpful advice and directions that he gave me to successfully complete the current thesis.

Special thanks, as well, to the other members of the three-member committee, Dr. Georgios Afendras and Dr. Katerina Hadjifotinou, for their presence, their teaching, and their concerns.

Last but not least, this project would not have been possible without the support of my parents, who have been by my side both financially and mentally, as well as my girlfriend Anna, who has shown great courage and patience, even in the darkest of hours, and whom I especially respect and admire.

Sapranidis Ioannis
Thessaloniki, Greece, January 2025

List of Figures

Some of the figures above were created with the help of AI and later processed by me.

List of Algorithms

List of Tables

Bibliography

[1] Mohammad Sadegh Es-haghi, Cosmin Anitescu, Timon Rabczuk, Methods for enabling real-time analysis in digital twins: A literature review, Computers and Structures, 2024, https://doi.org/10.1016/j.compstruc.2024.107342.

[2] Robert Johansson, Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib, Numerical Python, 978-1-4842-4246-9, 2019, https://doi.org/10.1007/978-1-4842-4246-9.

[3] Hans Petter Langtangen, Anders Logg, Solving PDEs in Python, The FEniCS Tutorial I, 978-3-319-52462-7, 2017, https://doi.org/10.1007/978-3-319-52462-7.

[4] Mats G. Larson, Fredrick Bengzon, The Finite Element Method: Theory, Implementation, and Applications, 978-3-642-33287-6, 2013, https://doi.org/10.1007/978-3-642-33287-6.

[5] Francisco Chinesta, Roland Keunings, Adrien Leygue, The Proper Generalized Decomposition for Advanced Numerical Simulations, A Primer, 978-3-319-02865-1, 2013, https://doi.org/10.1007/978-3-319-02865-1.

[6] Jonathan H. Tu, Clarence W. Rowley, Dirk M. Luchtenburg, Steven L. Brunton, J. Nathan Kutz, On Dynamic Mode Decomposition: Theory and Applications, arXiv:1312.0041, 2013, https://doi.org/10.48550/arXiv.1312.0041.

[7] Mark Asch, A Toolbox for Digital Twins: From Model-Based to Data-Driven, 978-1-61197-697-7, 2022, https://doi.org/10.1137/1.9781611976977.

[8] Fran¸ cois Cl´ement and Vincent Martin, The lax-milgram theorem. A detailed proof to be formalized in Coq, arXiv:1607.03618, 2016, https://doi.org/10.48550/arXiv.1607.03618.

[9] Peter Benner, Serkan Gugercin, and Karen Willcox, A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems, SIAM Review, 2017, https://doi.org/10.1137/130932715

[10] David L. Donoho, Matan Gavish, Elad Romanov, ScreeNOT: Exact MSE-Optimal Singular Value Thresholding in Correlated Noise, arXiv:2009.12297, 2023, https://doi.org/10.48550/arXiv.2009.12297.

[11] Rozza, G. (2014). Fundamentals of reduced basis method for problems governed by parametrized PDEs and applications. In: Chinesta, F., Ladevèze, P. (eds) Separated Representations and PGD-Based Model Reduction. CISM International Centre for Mechanical Sciences, vol 554. Springer, Vienna, https://doi.org/10.1007/978-3-7091-1794-1_4.

[12] Mario Ohlberger, Stephan Rave, Reduced Basis Methods: Success, Limitations and Future Challenges, arXiv:1511.02021, 2016, https://doi.org/10.48550/arXiv.1511.02021.

[13] B. O. Almroth, P. Stern and F. A. Brogan, Automatic choice of global shape functions in structural analysis, 2012, https://doi.org/10.2514/3.7539.

[14] Ahmed K. Noor, Recent advances in reduction methods for nonlinear problems, Computers & Structures, 2003, https://doi.org/10.1016/0045-7949(81)90106-1text.

[15] Chatterjee, Anindya, An introduction to the proper orthogonal decomposition, Current science, 2000, p 808-17.

[16] Julien Weiss, A Tutorial on the Proper Orthogonal Decomposition, 2019, https://doi.org/10.2514/6.2019-3333.

[17] Muruhan Rathinam and Linda R. Petzold, A New Look at Proper Orthogonal Decomposition, SIAM Journal on Numerical Analysis, 2003, https://doi.org/10.1137/S0036142901389049.

[18] S. Volkwein, Model Reduction using Proper Orthogonal Decomposition, Lecture notes, vol. 1025, 2011.

[19] Chinesta, F., Ladeveze, P. & Cueto, E. A Short Review on Model Order Reduction Based on Proper Generalized Decomposition. Arch Computat Methods Eng 18, 395–404 (2011), https://doi.org/10.1007/s11831-011-9064-7.

[20] Alberto Badías, David González, Iciar Alfaro, Francisco Chinesta, Elias Cueto, Local proper generalized decomposition,International Journal for Numerical Methods in Engineering, Volume 112, Issue 12, p. 1715-1732, 2017, https://doi.org/10.1002/nme.5578.

[21] Pierre Ladevèze, Ludovic Chamoin, On the verification of model reduction methods based on the proper generalized decomposition, Computer Methods in Applied Mechanics and Engineering, 2011, https://doi.org/10.1016/j.cma.2011.02.019.

[22] Tu, J. H., Dynamic mode decomposition: Theory and applications (Order No. 3597573), ProQuest Dissertations & Theses Global, 1458341928, https://www.proquest.com/dissertations-theses/dynamic-mode-decomposition-theory-applications/docview/1458341928/se-2?accountid=8359.

[23] Joshua L. Proctor, Steven L. Brunton, and J. Nathan Kutz, Dynamic Mode Decomposition with Control, SIAM Journal on Applied Dynamical SystemsVol. 15, Iss. 1 (2016)10.1137/15M1013857, 2016, https://doi.org/10.1137/15M1013857.

[24] P. J. Schmid, L. Li, O. Pust, Applications of the dynamic mode decomposition, Theoretical and Computational Fluid Dynamics, 25, 249–259, 2011, https://doi.org/10.1007/s00162-010-0203-9.

[25] Keller, J. B., Inverse Problems, The American Mathematical Monthly, 83(2), 107–118, 1976, https://doi.org/10.1080/00029890.1976.11994053.

[26] Richard C. Aster, Brian Borchers and Clifford H. Thurber, Parameter Estimation and Inverse Problems, 978-0-12-804651-7, 2018,https://doi.org/10.1016/C2015-0-02458-3.

[27] Curtis R. Vogel, Computational Methods for Inverse Problems, Frontiers in Applied Mathematics, 978-0-89871-757-0, 2002, https://doi.org/10.1137/1.9780898717570.

[28] Per Christian Hansen, Discrete Inverse Problems: Insight and Algorithms, Fundamentals of Algorithms, 978-0-89871-883-6, 2010, https://doi.org/10.1137/1.9780898718836.

[29] Jari P. Kaipio , Erkki Somersalo, Statistical and Computational Inverse Problems, 978-0-387-27132-3, 2006, https://doi.org/10.1007/b138659.

[30] Andreas Kirsch, An Introduction to the Mathematical Theory of Inverse Problems, 978-3-030-63343-1, 2021, https://doi.org/10.1007/978-3-030-63343-1.

[31] K. Ide, P. Courtier, A. Lorenc, Unified Notation for Data Assimilation : Operational, Sequential and Variational, Journal of the Meteorological Society of Japan, 1997, 10.2151/JMSJ1965.75.1B_181.

[32] Kody Law , Andrew Stuart , Konstantinos Zygalakis, Data Assimilation, A Mathematical Introduction, 978-3-319-20325-6, 2015, https://doi.org/10.1007/978-3-319-20325-6.

[33] Mark Asch, Marc Bocquet, and Maëlle Nodet, Data Assimilation: Methods, Algorithms, and Applications, Fundamentals of Algorithms, 978-1-61197-454-6, 2016, https://doi.org/10.1137/1.9781611974546.

[34] Wil Schilders, Introduction to Model Order Reduction, Model Order Reduction: Theory, Research Aspects and Applications, 2008, https://doi.org/10.1007/978-3-540-78841-6_1.

[35] Ram Kumar & D. Ezhilarasi, A state-of-the-art survey of model order reduction techniques for large-scale coupled dynamical systems, International Journal of Dynamics and Control, 2023, https://doi.org/10.1007/s40435-022-00985-7.

[36] Emad Samadiani, Yogendra Joshi, Reduced order thermal modeling of data centers via proper orthogonal decomposition: a review, International Journal of Numerical Methods for Heat & Fluid Flow, 2010, 10.1108/09615531011048231.

[37] Dogan Ibrahim, An Overview of Soft Computing, Procedia Computer Science, 2016, https://doi.org/10.1016/j.procs.2016.09.366.

[38] Vladimir Nasteski, An overview of the supervised machine learning methods, 2017, 10.20544/HORIZONS.B.04.1.17.P05.

[39] Jiachong Li, Regression and Classification in Supervised Learning, ICCBD '19: Proceedings of the 2nd International Conference on Computing and Big Data, 2019, https://doi.org/10.1145/3366650.336667

[40] Sen, P.C., Hajra, M., Ghosh, M., Supervised Classification Algorithms in Machine Learning: A Survey and Review, in: Mandal, J., Bhattacharya, D. (eds) Emerging Technology in Modelling and Graphics, Advances in Intelligent Systems and Computing, vol 937, Springer, Singapore, 2020, https://doi.org/10.1007/978-981-13-7403-6_11.

[41] R. Gentleman & V. J. Carey, Unsupervised Machine Learning, Bioconductor Case Studies, 2008, https://doi.org/10.1007/978-0-387-77240-0_10.

[42] Lorijn Zaadnoordijk, Tarek R. Besold & Rhodri Cusack, Lessons from infant learning for unsupervised machine learning, 2022, https://doi.org/10.1038/s42256-022-00488-2.

[43] Shengchun Wang, Hao Wang, Yunlai Zhou, Junbo Liu, Peng Dai, Xinyu Du, Magd Abdel Wahab, Automatic laser profile recognition and fast tracking for structured light measurement using deep learning and template matching, Measurement, 2021,https://doi.org/10.1016/j.measurement.2020.108362.

[44] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement Learning: A Survey, 1996, https://doi.org/10.1613/jair.301.

[45] Fengji Yi, Wenlong Fu, Huan Liang, Model-based reinforcement learning: A survey, 2018, https://api.semanticscholar.org/CorpusID:108339287.

[46] Setareh Rafatirad, Houman Homayoun, Zhiqian Chen, Sai Manoj Pudukotai Dinakarrao, Machine Learning for Computer Scientists and Data Analysts, From an Applied Perspective, 978-3-030-96756-7, 2022, https://doi.org/10.1007/978-3-030-96756-7.

[47] Christian Beck, Martin Hutzenthaler, Arnulf Jentzen, and Benno Kuckuck, An overview on deep learning-based approximation methods for partial differential equations, Discrete and Continuous Dynamical Systems - B, Volume 28, Issue 6: 3697-3746, 2023, 10.3934/dcdsb.2022238.

[48] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, 2019, https://doi.org/10.1016/j.jcp.2018.10.045.

[49] Konstantinos Prantikos, Lefteri H. Tsoukalas, and Alexander Heifetz, Physics-Informed Neural Network Solution of Point Kinetics Equations for a Nuclear Reactor Digital Twin, 2022, https://doi.org/10.3390/en15207697.

[50] Vishal Jadhav, Anirudh Deodhar, Ashit Gupta, Venkataramana Runkana, Physics Informed Neural Network for Health Monitoring of an Air Preheater, Vol. 7 No. 1 (2022): Proceed-

ings of the European Conference of the PHM Society, 2022, https://doi.org/10.36001/phme.2022.v7i1.3343.

[51] Karan Taneja, Xiaolong He, QiZhi He, Xinlun Zhao, Yun-An Lin, Kenneth J. Loh, Jiun-Shyan Chen, A Feature-Encoded Physics-Informed Parameter Identification Neural Network for Musculoskeletal Systems, J Biomech Eng. Dec 2022, 144(12): 121006, https://doi.org/10.1115/1.4055238.

[52] E. Samaniego, C. Anitescu, S. Goswami, V.M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Computer Methods in Applied Mechanics and Engineering, 2020, https://doi.org/10.1016/j.cma.2019.112790.

[53] Vien Minh Nguyen-Thanh, Xiaoying Zhuang, Timon Rabczuk, A deep energy method for finite deformation hyperelasticity, European Journal of Mechanics - A/Solids, 2020, https://doi.org/10.1016/j.euromechsol.2019.103874.

[54] A. M. Vinogradov, B. A. Kupershmidt, The structures of Hamiltonian mechanics, Russian Mathematical Surveys, 1977, Volume 32, Issue 4, Pages 177–243, https://doi.org/10.1070/RM1977v032n04ABEH001642.

[55] Cowley, R. A., Structural phase transitions I. Landau theory, Advances in Physics, 29(1), 1–110, 1980, https://doi.org/10.1080/00018738000101346.

[56] Gomez, Hector; van der Zee, Kristoffer George, Computational phase-field modeling. In E. Stein, R. de Borst, & T. J. Hughes (Eds.), Encyclopedia of Computational Mechanics, 2017, https://nottingham-repository.worktribe.com/output/897296.

[57] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs, 2023, http://jmlr.org/papers/v24/21-1524.html.

[58] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Physics-Informed Neural Operator for Learning Partial Differential Equations, ACM/JMS Journal of Data Science, Volume 1, Issue 3, Article No.: 9, Pages 1 - 27, 2024, https://doi.org/10.1145/3648506.

[59] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Fourier Neural Operator for Parametric Partial Differential Equations, arXiv:2010.08895, 2021, https://doi.org/10.48550/arXiv.2010.08895.

[60] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, Markov Neural Operators for Learning Chaotic Systems, arXiv:2106.06898, 2022, https://doi.org/10.48550/arXiv.2106.06898.

[61] Qianying Cao, Somdatta Goswami, George Em Karniadakis, LNO: Laplace Neural Operator for Solving Differential Equations, arXiv:2303.10528, 2023, https://doi.org/10.48550/arXiv.2303.10528.

[62] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang & George Em Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nature Machine Intelligence volume 3, pages218–229, 2021, https://doi.org/10.1038/s42256-021-00302-5.

[63] Tapas Tripura, Souvik Chakraborty, Wavelet neural operator: a neural operator for parametric partial differential equations, arXiv:2205.02191, 2022, https://doi.org/10.48550/arXiv.2205.02191.

[64] Hubert Baty, A hands-on introduction to Physics-Informed Neural Networks for solving partial differential equations with benchmark tests taken from astrophysics and plasma physics, 2024, hal-04491808, https://hal.science/hal-04491808v1.

[65] Joachim Schoeberl, NGS-Py Documentation, Release 6.2, 2018

[66] Kuan-Chung Lin, Kuo-Chou Wang, Cheng-Hung Hu, Investigating deep energy method applications in thermoelasticity, Engineering Analysis with Boundary Elements, Volume 159, February 2024, Pages 302-314, https://doi.org/10.1016/j.enganabound.2023.12.012.

[67] Sifan Wang, Hanwen Wang, Paris Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets, 2021, https://doi.org/10.48550/arXiv.2103.10974.

[68] Vien Minh Nguyen-Thanh, Xiaoying Zhuang, Timon Rabczuk, A deep energy method for finite deformation hyperelasticity, European Journal of Mechanics - A/Solids, Volume 80, 2020, https://doi.org/10.1016/j.euromechsol.2019.103874.

[69] Vien Minh Nguyen-Thanh, Cosmin Anitescu, Naif Alajlan, Timon Rabczuk, Xiaoying Zhuang, Parametric deep energy approach for elasticity accounting for strain gradient effects, Computer Methods in Applied Mechanics and Engineering, Volume 386, 2021, https://doi.org/10.1016/j.cma.2021.114096.

[70] Léon Bottou, Frank E. Curtis, Jorge Nocedal, Optimization Methods for Large-Scale Machine Learning, arXiv:1606.04838, 2018, https://doi.org/10.48550/arXiv.1606.04838.