# POLITECNICO DI TORINO

**Master's Degree in Mathematical engineering**


**Master's Degree Thesis**

# Physics Informed Neural Networks and Neural Tangent Kernel: preliminary results for parametric Optimal Control Problems

**Supervisors**

**Ph.D Maria STRAZZULLO**

**Ph.D Federico PICHI**

**Prof. Gianluigi ROZZA**

**Candidate**

**Andrea TATARANNI**

**November 2024**

# Acknowledgements

*Ringrazio i miei genitori e mio fratello Luca che mi hanno dato la preziosa serenità nello studio durante questi anni, il mio migliore amico Alessandro per i mai scontati consigli, ma soprattutto la mia ragazza Silvia per essermi stata sempre e incessantemente vicino questi cinque anni in cui siamo cresciuti insieme. Ringrazio infine la Dott.ssa Maria Strazzullo, il Dottor Federico Pichi e il Prof. Gianluigi Rozza per il prezioso aiuto durante tutta l'attività di tesi.*

# Acronyms

**OCP**
Optimal Control Problem

**PINN**
Physics-informed Neural Networks

**PIARCH**
Physics-informed Architecture

**PDE**
Partial Differential Equation

**NTK**
Neural Tangent Kernel

**SB**
Spectral Bias

**FEM**
Finite Element Method

**BC**
Boundary conditions

**FFE**
Fourier Feature Embedding

**NN**
Neural Network

# Abstract

Physics-Informed Neural Networks (PINNs) offer a promising framework for solving differential problems, including Partial Differential Equations (PDEs) and Optimal Control Problems (OCPs). They have also been explored in parametric settings, which involves PDEs that depend on a set of parameters. In this context, the objective is to create a framework capable of efficiently generating numerical approximations of the solution when the parameter input of the differential problem changes. In fact, standard numerical methods can be too time-consuming to solve the problem in real-time and for many parameters. This thesis focuses on applying PINNs parametric OCPs. The first contribution is the improvement of the performances of standard PINNs on two test cases already investigated in the literature: a parametric Elliptic OCP and a parametric Stokes OCP. Improvements have been achieved by studying different aspects such as the sampling techniques, the use of an alternative architecture, named PIARCH, and the strong enforcing of the Dirichlet boundary conditions. Although results should still be improved, we enhanced the performance, for both the problems. To better study the training phase capabilities, we focused on the theory of Neural Tangent Kernel (NTK), which is a matrix that describes how the training of a generic neural network evolves. The theory behind NTK helps explain why neural networks, and specifically PINNs, often fail to train. It has been proven that, while neural networks can easily learn low-frequency components of the training data, they struggle to learn higher-frequency components, a challenge known as Spectral Bias. To address this issue, we study two approaches: augmenting the input of the PINN and applying an adaptive balancing of the loss function. After evaluating these approaches on one-dimensional uncontrolled problems, we applied them to two parametric OCPs we investigated to improve their performances.

# Chapters description

**Problem formulation and discretization** In this chapter the OCPs and parametric OCPs formulations are introduced along with some key theoretical aspects. It also provides a basic explanation of the Galerkin method.

**Physics-informed Neural Networks** After introducing Neural Networks, it discusses the general pipeline of their working procedure, with a special focus on Physics-Informed Neural Networks. Afterwards, it describes Neural Tangent Kernel and its implications.

**Practical insights on Fourier Embeddings and Adaptive weights** In this chapter, one-dimensional simple problems are solved with the two proposed techniques.

**PINNs for Optimal Control Problems** In this chapter we solve two proposed parametric Optimal Control Problems, parametric Elliptic Optimal Control and parametric Stokes Optimal Control, studying the sensitivity of the PINN with respect to some features.

**Fourier Embedding and Adaptive weights for Optimal Control Problems** | In this section we apply the two approaches, i.e. Fourier Embedding and Adaptive weights, to the best setting found in the previous chapter.

# Table of Contents

# Chapter 1

# Introduction

Optimal Control problems (OCP) are differential models with a wide range of applications in engineering and sciences such as fluid dynamics [1, 2, 3], more specifically environmental sciences [4] and hemodynamic [5], but also electronics [6]. The model consists in a cost functional that has to be minimized subject to constraints which often represent the governing Partial Differential Equations (PDE) of the physical model (for example the Navier-Stokes equation in hemodynamic). The cost functional usually represents a desired configuration of the model that has to be reached. For example in [4] the purpose was to maintain under a certain level the concentration of a pollutant in a specific region of the sea, in Trieste region, Italy.
In many applications, OCPs depend also on a set of parameters. This leads to a more complex model which is hard to solve efficiently with traditional discretisations, suchs as Finite Element Method (FEM). An efficient way of dealing with this kind of problem could be by using Model Order Reduction (MOR), which have been deeply studied in publications such as [7, 8, 9, 10] and, more specifically for OCPs MOR is a technique used to speed up simulations, allowing for a more efficient analysis of the relationship between a model's inputs and outputs. By simplifying complex models while retaining essential dynamics, MOR enables faster computations, which facilitates a smoother and more comprehensive study of system behavior under various input conditions.

There are numerous methods available for performing Model Order Reduction (MOR) (see [11] for a reference), including techniques that create a reduced set of simulations to serve as basis functions, enabling faster future simulations within the finite element framework. However, this thesis focuses on a more specialized approach based on machine learning: Physics-Informed Neural Networks (PINNs) for parametric Partial Differential Equations (PDEs).

PINNs are a recent method for computing numerical approximations of PDEs, based on Neural Networks (NN). They work by incorporating the physical information into a NN simply by adding the residual of the PDE in the training phase. They were introduced in [12] and their usage spread during the last years, since they exhibited the possibility of solving PDEs in relevant practical application such as [13, 14, 15, 16, 17, 18, 19, 20]. Other positive features of PINNs rely on their mesh-free approach and on the possibility of solving high dimensional PDEs, which is impossible for FEM.
Despite promising results, PINNs struggle in approximating multi-scale problems and converges towards low-frequencies solutions. This problem is known in literature as Spectral Bias (SB) and it is well documented in [21, 22, 23]. The Neural Tangent Kernel (NTK) is a kernel describing the evolution of the training of the NN, it can be computed as the derivative of the residuals of the network with respect to the parameters of the model, and its properties are the most addressed for explaining SB. In order to fix this behavior many solutions have been proposed, but in this thesis we will focus on testing two possible approaches for parametric OCPs:

- The first one is called Fourier Feature Embedding (FFE). The FFE consists in expanding the input data into an higher dimensional space using trigonometric functions. This helps the PINN in gaining more expressivity making it easier for the NN to learn detailed and oscillatory features in the data, resulting in faster training and improved accuracy. This procedure was introduced for general deep learning tasks in [24], but then applied also for PINN in [25] showing promising results.

- The second approach involves the using of adaptive weights in the loss function of the PINN, following the algorithm proposed in [21], we studied this approach in relation to OCPs, in order to see if it improves the accuracy also for parametric OCPs. The idea behind is strictly related to NTK theory and will be better explained in 3.4.

Concerning the experiments, we studied two problems taken from [26], the first is an OCP constrained to a Poisson problem, the second a constrained to Stokes equation. Firstly, we enhanced the performances of the PINN by training different settings and understanding what were the best features and structure to use. Afterwards we implemented FFE and NTK in order to investigate if one or both help in enhancing the training of the PINN and its accuracy.

The work behind the writing of this thesis has been divided between Politecnico di Torino (PoliTO) and Scuola Internazionale degli Studi Superiori Avanzati (SISSA). For the practical aspects of this thesis, the libraries PINA and RBniCS were used,

the first concerning PINNs and the second used for the finite elements simulations. Both are available at the SISSA mathLab GitHub website.

# Chapter 2

# Problem formulation and discretization

In this section, we will introduce what OCPs, then we will see their generalization to parametric OCP and some known methods to solve them. Afterwards, we will introduce the Galerkin discretization related to FEM.

## 2.1 Optimal control problems

The main features of an Optimal Control Problem are schematically presented in Figure 2.1. The system is described by a *state* variable $y$, a variable $u$ called *control* that represents something we can regulate and an *observable* of interest $p$ that is typically derived from the state variable. Usually $y$ is the solution of a generic differential problem such as an ordinary differential equation or a PDE. The aim of an OCP is to find the right control $u$ in order to make the system reach a desirable value of the observable of interest $p_d$.

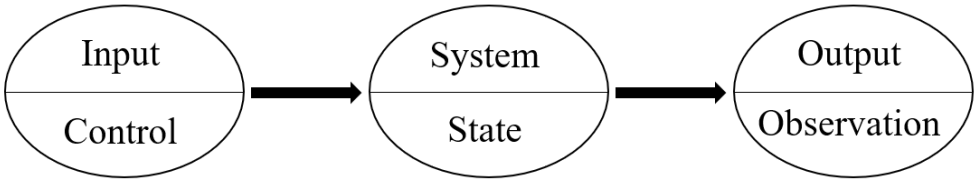



**Figure 2.1:** Basic scheme of an OCP, adapted from [27].

Let us now introduce the formalism of an OCP. Let $U$, $Y$ and $Z$ be suitable Banach spaces and $U_a \subset U$ and $Y_a \subset Y$ the admissible spaces for the control variable and the state variable respectively. Then we can consider the following OCP:

$$\begin{aligned} &\min_{(y,u)\in(U_a,Y_a)} J(y,u) \\ &\text{s.t. } \mathcal{E}(y,u)=0, \end{aligned} \tag{2.1}$$

where $J : Y \times U \to \mathbb{R}$ is a functional and $\mathcal{E} : Y \times U \to Z$ represents the state equation. One of the most important aspects when investigating mathematical models is to understand if the model admits a solution and under which assumptions. For this reason we introduce the following theorem:

**Theorem 1** *Let us assume as true the following hypothesis:*

*i.* $U_a$ *is convex, bounded and closed;*

*ii.* $Y_a$ *is convex and closed such that* (2.1) *has a feasible point;*

*iii. the state equation* $\mathcal{E}(y,u)$ *has a bounded solution operator* $u \in U_a \to y(u) \in Y$*;*

*iv.* $\mathcal{E} : Y \times U \to Z$ *is continuous under weak convergence;*

*v.* $J$ *is weakly lower semicontinuous;*

*then the problem* (2.1) *has an optimal solution* $(\bar{y}, \bar{u})$*.*

The OCP's cost function that we are going to take into account can be written in the following form

$$J(y(u), u, p(y(u)); p_d) = \frac{1}{2}\|p(y(u)) - p_d\|_P^2 + \frac{\alpha}{2}\|u\|_U^2, \tag{2.2}$$

where $\alpha > 0$ and $p : Y \to P$. But practically we will see only the case where $p \equiv y$. We can also relieve the notation of $y(u)$ to only $y$, therefore:

$$J(y, u; y_d) = \frac{1}{2}\|y - y_d\|_Z^2 + \frac{\alpha}{2}\|u\|_U^2. \tag{2.3}$$

The parameter $\alpha$ is an important penalization term. In fact $\alpha$ is needed to penalize control functions that are too costly. Higher values of $\alpha$ enforce the request of having a "small" (in norm) control. If we try to solve the OCP without it, we could get an unrealistic control with high peaks and variation, which cannot be reached in a concrete setting, but matematically we lose the uniqueness of the solution.

## 2.2 Parametric Optimal Control Problem

Let us now introduce the reduced problem. Always accounting the problem (2.1), if we assume that $J$ and $\mathcal{E}$ are continuously Fréchet differentiable and that for every control $u$ the state equation $\mathcal{E}$ admits a solution $y(u)$ then we can define a solution operator $K : u \in U \to y(u) \in Y$. For these reason we can re-define the OCP's functional $J$ making it depending only on the control variable $u$, because if $u$ is defined, for $y$ is the same thanks to the operator $K$. The new OCP reduced problem is:

$$\begin{aligned} &\min_u \hat{J}(u) = J(y(u), u) \\ &\text{s.t. } \; u \in \hat{U}_a = \{u \in U : (y(u), u) \in Y \times U\}. \end{aligned} \tag{2.4}$$

This formulation is important in order to demonstrate the following

**Theorem 2** *Assume that*

*i.* $U_a \subset U$ *is nonempty, closed and convex;*

*ii.* $J : Y \times U \to \mathbb{R}$ *and* $\mathcal{E} : Y \times U \to Z$ *are continuously Fréchet differentiable and* $U, Y, Z$ *are Banach spaces*

*iii. if* $V \subset U$ *neighborhood of* $U_a$*, the state equation* $\mathcal{E}(y, u) = 0$ *has a unique solution* $y = y(u) \in Y$*;*

*iv.* $\mathcal{E}_y(y(u), u)$ *has a bounded inverse for all* $u \in V \supset U_a$*;*

*then, if* $\bar{u}$ *is a local solution of* (2.4)*,* $\bar{u}$ *also satisfies the following variational inequality*

$$\langle \hat{J}'(\bar{u}), v - \bar{u} \rangle_{U^*,U} \geq 0 \;\; \forall v \in U_a. \tag{2.5}$$

The notation $U^*$ stands for the dual space of $U$. Equation (2) is also known as Optimality Condition. Let us now define the Lagrangian functional $\mathcal{L} : Y \times U \times Z^* \to \mathbb{R}$ as

$$\mathcal{L}(y, u, z) = J(y(u), u) + \langle z, \mathcal{E}(y, u) \rangle_{Z^*,Z}, \tag{2.6}$$

where $z \in Z^*$ is the adjoint variable also known as Lagrange Multiplier. If we can compute an explicit formulation of $\hat{J}'(\bar{u})$ we can use (2) plus the definition of the Lagrangian (2.6) to prove the following corollary for OCPs:

**Corollary 1** *With the same hypothesis of theorem* (2) *plus having* $U_a \equiv U$*, if* $(\bar{y}, \bar{u})$ *is an optimal solution of the problem* (2.1)*, then there exists an adjoint state (also known as Lagrange multiplier)* $\bar{z} \in Z^*$ *such that the following equations hold true*

$$\begin{cases} \langle D_y \mathcal{L}(\bar{y}, \bar{u}, \bar{z}), v \rangle = 0, & \forall v \in Y, \\ \langle D_u \mathcal{L}(\bar{y}, \bar{u}, \bar{z}), \omega \rangle = 0, & \forall \omega \in U, \\ \langle D_z \mathcal{L}(\bar{y}, \bar{u}, \bar{z}), q \rangle = 0, & \forall q \in Y. \end{cases} \tag{2.7}$$

The system (2.7) is also known as Optimality System. All details about the previous theorems can be found in [28, Sections 1.5-1.6].

Let us now introduce parametric OCPs: they can be easily seen as a generalization of a standard OCP with the only difference that the problem depends also on a set of parameters $\boldsymbol{\mu}$:

$$\begin{aligned} &\min_{(y(\boldsymbol{\mu}), u(\boldsymbol{\mu})) \in (Y_a, U_a)} J(y(\boldsymbol{\mu}), u(\boldsymbol{\mu}); \boldsymbol{\mu}) \\ &\text{s.t. } \mathcal{E}(y(\boldsymbol{\mu}), u(\boldsymbol{\mu}); \boldsymbol{\mu}) = 0 \text{ in } \Omega, \end{aligned} \tag{2.8}$$

where $\boldsymbol{\mu} \in \mathbb{P} \subset \mathbb{R}^D$ is a set of $D$ parameters.
All the results developed earlier are also true for parametric OCPs, especially focusing on the Lagrange system (2.7).
This setting provides the real challenge of this work: the goal will be solving the parametric OCP with only one neural network that can give the solution with respect to any input parameters. The NN will not learn the solution $(y(\mathbf{x}, \boldsymbol{\mu}), u(\mathbf{x}, \boldsymbol{\mu}), z(\mathbf{x}, \boldsymbol{\mu}))$ of the state variable, control variable and adjoint variable for a fixed $\boldsymbol{\mu}$, but all the solutions with respect to the parametric domain $\mathcal{P}$. Thus $\boldsymbol{\mu}$ will be a input of the framework.
If we define $M : \boldsymbol{\mu} \in \mathcal{P} \to (y(\mathbf{x}, \boldsymbol{\mu}), u(\mathbf{x}, \boldsymbol{\mu}), z(\mathbf{x}, \boldsymbol{\mu})) \in (Y, U, Y)$ then the space

$$\mathcal{M} := \{M(\boldsymbol{\mu}) = (y(\mathbf{x}, \boldsymbol{\mu}), u(\mathbf{x}, \boldsymbol{\mu}), z(\mathbf{x}, \boldsymbol{\mu})) : \boldsymbol{\mu} \in \mathcal{P}\}, \tag{2.9}$$

that is called Solution Manifold, contains all the possible solutions of the OCP when the parameters $\boldsymbol{\mu}$ belong to $\mathcal{P}$. The map $M$ represent the function that we want to surrogate with a NN.

The previous task is suited in the MOR setting, because the real problem does not consist in finding a solution of the problem fixed a parameter, which, in theory, could be an easy task achievable with methods such as FEM, but to generate a framework that can quickly compute a new solution given a parameter $\boldsymbol{\mu}^* \in \mathcal{P}$. In fact, for instance, FEM could take too much time to run, making them not suitable for real time applications. In the case of PINNs the gain is that the training phase

is achieved during an "offline" phase, after that it is extremely fast to get a solution using the pre-trained network. A more interested reader may refer to [11] for a more detailed explanation of MOR-based methods.

For completeness we briefly introduce what are the classical methods of solving OCPs. Here some hints will be given, but more details are given, for example in [27]. In general there are two pradigms that can be used to solve differential problems: optimize then discretize, discretize then optimize. Let us suppose to reformulate the problem (2.1) into

$$\text{find } u \in U \text{ such that } J(u) < J(v) \quad \forall v \in U, \tag{2.10}$$

then, two approaches can be formulated as follows:

**1) Discretize then optimize** Firstly we discretize the control variable space $U_a$ into $U_{a,h}$ and the state equation accordingly to the finite element theory, obtaining:

$$\mathcal{E}_h(y_h, u_h) = 0, \tag{2.11}$$

then we search for a discretized control $u_h$ such that

$$J(y_h(u_h), u_h) < J(y(v_h), v_h) \;\; \forall v_h \in U_{a,h}. \tag{2.12}$$

**2) Optimize then discretize** This approach is quite different. Firstly we have to use Lagrange optimality condition to write the system

$$\mathcal{G}(y, u, z) = 0, \tag{2.13}$$

where $\mathcal{G}$ is only a compact way of referring to the system (2.7). Then we can discretize this problem and solve it numerically. This was the approach that was used to compare the results with respect to the high fidelity solution, which, for us, will be the finite elements one. This choice allows us to compare also the accuracy of the PINN with respect to the adjoint variables, although the major interest will be on physical variables, i.e. state and control.
The two described approaches do not consistently yield identical results. For instance in [29], where the state equation deals with a dynamic problem, applying a finite element approximation can result in discrepancy because the method is not precise enough for high-frequency solutions.

## 2.3 Galerkin approximation

In this section we briefly introduce and describe some basic aspects about the Galerkin approximation which is used by FEM, discretization on which we relied to produce the high fidelity simulations used as a comparison for the PINN ones.

Let us take a generic elliptic problem enforced with Dirichlet boundary conditions:

$$\begin{cases} \mathcal{A}(u) = f(x) \quad \text{in} \quad \Omega, \\ u = g_D \quad \text{on} \quad \Gamma_D, \end{cases} \tag{2.14}$$

such that $\Gamma_D \subseteq \partial\Omega$. The elliptic operator is represented by $\mathcal{A}$. It is well known [27] that this problem can be reformulated in variational form: find $u \in V$ such that

$$a(u, v) = F(v), \quad \forall v \in V, \tag{2.15}$$

where $V$ is a suitable Hilbert space subset of the Sobolev space $H^1$. If $a : V \times V \to \mathbb{R}$ is a continuous and coercive bilinear form, and if $F : V \to R$ is a continuous and linear form, then, for the Lax-Milgram lemma, the problem (2.15) admits a unique solution.
Let us take a space $V_h \subset V$ such that:

$$\dim(V_h) = N_h < \dim(V) = \infty, \tag{2.16}$$

where $h > 0$ is an index related to the dimension of $V_h$: $N_h$ increases as $h$ shrinks. Suppose to solve the same variational problem of before using $V_h$: find $u_h \in V_h$ such that

$$a(u_h, v_h) = F(v_h), \quad \forall v_h \in V_h. \tag{2.17}$$

The problem (2.17) is also known as Galerkin Problem. It easy to prove, always with Lax-Milgram theorem, that it admits a unique solution. If we take a basis $\{\phi_j\}_{j=1}^{N_h}$ of the space $V_h$, we can describe every function of the space with respect to the basis we have just taken as

$$u_h(\mathbf{x}) = \sum_{j=1}^{N_h} \phi_j(\mathbf{x}) u_j, \tag{2.18}$$

where $u_j$ are the components of $u_h$ with respect to the basis $\{\phi_j\}_{j=1}^{N_h}$. If we insert (2.18) into (2.17) and take tests functions $v_h = \phi_k$ we obtain the following expression:

$$
\begin{aligned}
&a(\sum_{j=1}^{N_h} \phi_j(\mathbf{x})u_j, \phi_k) = F(\phi_k) \quad j = 1...N_h,\ k = 1...N_h, \\
&\sum_{j=1}^{N_h} u_j a(\phi_j, \phi_k) = F(\phi_k) \quad j = 1...N_h,\ k = 1...N_h.
\end{aligned}
\tag{2.19}
$$

Now, we define the matrix $A = (a_{i,j} = a(\phi_j, \phi_k))$, the vector $\boldsymbol{f} = (f_k = F(\phi_k))$ and the vector $\boldsymbol{u} = (u_j)$, thus the second equation in (2.19) can be written as a linear system

$$A\boldsymbol{u} = \boldsymbol{f}, \tag{2.20}$$

that can be solved in order to obtain the vector $\boldsymbol{u}$, i.e. the components of the solution to the Galerkin problem with respect to the basis $\{\phi_j\}_{j=1}^{N_h}$. In this framework $u_h$ is exactly the function that approximates the solution of the elliptic problem.

One of the most important results about the Galerkin method is the Céa Lemma:

**Theorem 3** *If $a : V \times V \to \mathbb{R}$ is a bilinear application continuous with constant $C$ and coercive with constant $\alpha$, then referring to the problems* (2.17) *and* (2.15)*, the following inequality holds:*

$$||u - u_h||_V \leq \frac{C}{\alpha} \inf_{w_h \in V_h} ||u - w_h||_V. \tag{2.21}$$

The Lemma tells us that the convergence of the method is always reached until we satisfy the following condition:

$$\lim_{h \to 0} ||v - v_h||_V, \tag{2.22}$$

that can be intended as a request of $V_h$ tending to "fill" the space $V$ as $h$ approaches 0.

The procedure just described is at the basis for the FEM that we used in this context to obtain a high fidelity solution, in order to compare the results of the PINN when we had not a ground truth solution of the problem. In particular we used the plynomials of degree 1 as basis functions for writing the Galerkin system. All the theory we developed in this section is naturally extendable to OCPs.

# Chapter 3

# Physics-informed Neural Networks

In this section, we will introduce what Neural Networks is, then we will move toward the standard definition of a PINN. Afterwards the NTK and the SB will be described, along with Fourier Feature embedding.

## 3.1 Introduction to PINN

Physics-Informed NN are a recent research area that developed thanks to the advancing of machine learning techniques and to the rapid growth of computational resources. They can both be applied to direct and inverse problems as shown in [30] and [31]. For solving direct problems, PINNs take a different approach compared to classical NN. Instead of learning from empirical datasets, PINNs use the physical laws described by PDEs as their source of information. In fact by sampling points within the problem domain, it is possibile to calculate the residual of the PDE through automatic differentiation. Then the loss can be computed and be minimized, guiding the network to solutions consistent with the governing physics.
This tool was introduced in [12] and its study has significantly expanded over the past years. The reason is that the standard methods that are used for numerical approximation such as finite elements, finite differences and spectral methods suffer from the "curse of dimensionality", which means that the approximation is infeasible when the PDE involves many dimensions due to the lack of computational power. In fact, when the dimension of the PDE increases, the computational effort needed increases.
Many relevant problems, rapidly chemistry, economy and physics, involve the

formulation of a PDE with high dimensional spaces, i.e. the Black Scholes equations for pricing the derivatives and the Schrödinger equation for many body problems.

## 3.2 Neural Networks

A neural network is a non-linear function approximator that can be used to find relationship between data. It can be modeled, as the name suggests, as the neurons of the human brain. Every neuron takes the information and propagates it, concerning FeedForward Neural Networks, to all other neurons of the following layer of neurons as shown in the scheme of Figure 3.1. The function within the neurons is:

$$N_j^k(\boldsymbol{x}) = \sigma\left(\sum_{i=1}^{m} x_i \omega_i^{jk} + \omega_0^{jk}\right), \tag{3.1}$$

where $\omega_i^{jk} \in \mathbb{R}$ are trainable parameters of the neuron in $k$-th layer at $j$-th position, the parameter $\omega_0^{jk}$ is usually known as "bias", $m$ is the number of data taken from the previous layer, while $x_i$ is the information taken from the previous neurons. The function $\sigma$ is known as "activation function", it is usually non-linear. The first and the last layer of the network are the input layer and the output layer respectively, the first one takes as input the data, while the output layer, usually, is a scalar that represents the label predicted with respect to the input taken. For example, always addressing to Figure 3.1, we have that $\boldsymbol{z} = [N_1^1(x), N_2^1(x), N_3^1(x)]$ and $\boldsymbol{t} = [N_1^2(\boldsymbol{z}), N_2^2(\boldsymbol{z}), N_3^2(\boldsymbol{z})]$, but $\boldsymbol{t}$ can also be written recursively with respect to $x$:

$$\begin{aligned} \boldsymbol{t} = [&N_1^2([N_1^1(x), N_2^1(x), N_3^1(x)]), \\ &N_2^2([N_1^1(x), N_2^1(x), N_3^1(x)]), \\ &N_3^2([N_1^1(x), N_2^1(x), N_3^1(x))]]. \end{aligned} \tag{3.2}$$

And this is just with a very modest Neural Network. It is clear that bigger networks work with a very complex structure that is difficult to interpret.

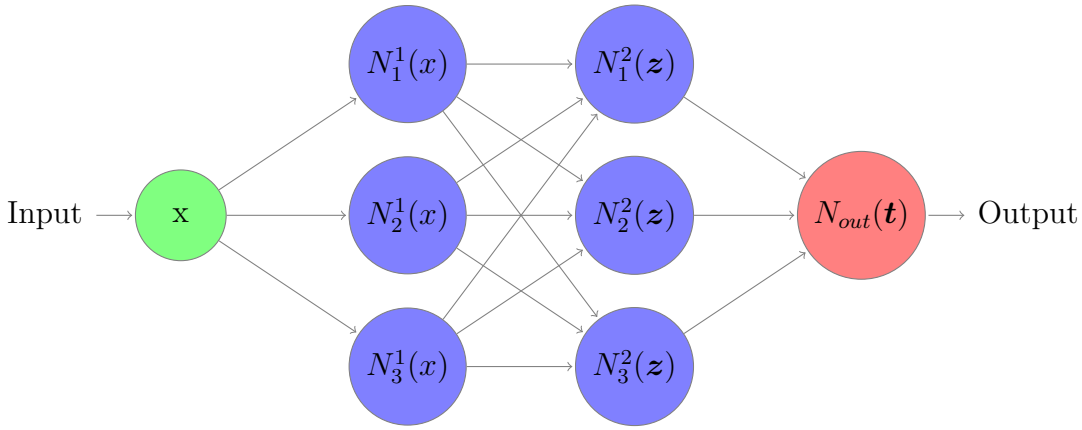



**Figure 3.1:** Simple NN with 2 hidden layers and 3 neurons for each of them, with one input and one output.

In this framework, if we have a training set $T = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ of $N$ examples, which represents the couple of predictors and labels, we can easily define a loss function that measures how much the approximation of the neural networks fails the predictions on the training set. If $\boldsymbol{x}_i \in \mathbb{R}^n$ and $y_i$ is a scalar, then we can define $\tilde{f}_{\boldsymbol{\omega}} : \mathbb{R}^n \to \mathbb{R}$ as the function represented by the NN, which depends also on the parameters $\boldsymbol{\omega}$. The loss function will be

$$L_Q(\boldsymbol{\omega}) = ||y - \tilde{f}_{\boldsymbol{\omega}}(\boldsymbol{x})||_Q, \tag{3.3}$$

where $Q$ is a norm computed using the set $T$ and $\tilde{f}$ is the surrogate function of the NN. Many loss functions can be defined, as for every task there may be an appropriate one, although in this work we will not explore them and use the most common one that we will introduce later.
Now we will remind some known theoretical results that let us understand the power of NNs: the universal approximation theorems. The following theorems are taken from [32].

**Theorem 4** *Let $f : K \subset \mathbb{R}^n \to \mathbb{R}^m$ be a function with $K$ compact and $f \in C^0(K, \mathbb{R}^m)$. Let $\tilde{f} : K \to \mathbb{R}^m$ be the NN function and $\sigma : \mathbb{R} \to \mathbb{R}$ its activation function. Then*

$$\forall n, \forall m, \forall K \subset \mathbb{R}^n, \forall f \in C^0(K, \mathbb{R}^m), \forall \epsilon > 0,$$
$$\exists d \in \mathbb{N}, W^1 \in \mathbb{R}^{d \times n}, \boldsymbol{\omega}_0 \in \mathbb{R}^d, W^2 \in \mathbb{R}^{m \times d} \quad \textit{such that}$$
$$\boldsymbol{y} = f(x) = W^2(\sigma(W^1\boldsymbol{x} + \boldsymbol{\omega}_0)) \Rightarrow ||\tilde{f} - f||_{\infty,K} < \epsilon, \iff \sigma \quad \textit{is not polynomial}$$

The previous theorem states that a NN, with just one hidden layer, can approximate every function that respect the hypothesis with an error at will provided the right number of neurons in that layer.

**Theorem 5** *Suppose $\sigma : \mathbb{R} \to \mathbb{R}$ be a non linear, continuously differentiable in at least one $x \in \mathbb{R}$, where $\sigma(x) \neq 0$, activation function. Then*

$$\forall n, \forall m, \forall K \subset \mathbb{R}^n, \forall f \in \mathbb{C}^0(K, \mathbb{R}^m), \forall \epsilon > 0$$
$$\exists L > 0, \exists NN \text{ with } L \text{ layers each with } n+m+2 \text{ neurons} \mid ||\tilde{f} - f||_{\infty,K} < \epsilon.$$

This theorem states that for similar hypothesis if we increase the number of layers and we fix the number of neurons we can always get an error at will.

**Theorem 6** *For every interval $[a, b]$ there exists $\sigma : \mathbb{R} \to \mathbb{R}$ continuous and computable such that*

$$\forall n, \forall f \in \mathbb{C}^0([a,b]^n, \mathbb{R}) \exists NN \text{ with } n \text{ neurons in the first layer}$$
$$2n+2 \text{ neurons in the second layer} \mid ||\tilde{f} - f||_{[a,b]^n,\infty} < \epsilon.$$

The previous theorems show us how powerful NN can be. Unfortunately, they do not give us any information on how many neurons or layers should we use, when we are approaching a specific task.
We now introduce the PINNs more in depth, explaining their functioning and, later on, some improvements that have been made until today.

## 3.3 PINN formulation

Let us take a generic 2-dimensional PDE with generic Neumann and Dirichlet boundary conditions:

$$\begin{cases} \mathcal{A}[u](x,y) = \mathcal{F}(x,y), & \text{in } \Omega, \\ u(x,y) = g_D(x,y), & \text{on } \Gamma_D, \\ \frac{\partial u}{\partial \boldsymbol{n}}(x,y) = g_N(x,y), & \text{on } \Gamma_N, \end{cases} \tag{3.4}$$

where $\Gamma_D$, $\Gamma_N$ represent the boundaries enforced with Dirichlet and Neumann conditions respectively, such that $\{\Gamma_D, \Gamma_N\}$ is a partition of $\partial\Omega$. The vector $\boldsymbol{n}$ is a vector orthogonal to $\Gamma_N$ and the functions $g_D : \mathbb{R}^2 \to \mathbb{R}$ and $g_N\mathbb{R}^2 \to \mathbb{R}$ are the information on the boundaries. The symbol $\mathcal{A}$ represents a generic differential problem, for instance it could be the Stokes equation or the heat equation, while $\mathcal{F}$ is the forcing term. The residuals of this problem are:

$$\begin{aligned} &\mathcal{R}_{pde}(x,y) = \mathcal{A}[u](x,y) - \mathcal{F}(x,y), \quad \text{for } (x,y) \in \Omega, \\ &\mathcal{R}_D(x,y) = u(x,y) - g_D(x,y), \quad \text{for } (x,y) \in \Gamma_D, \\ &\mathcal{R}_N(x,y) = u(x,y) - g_N(x,y), \quad \text{for } (x,y) \in \Gamma_N. \end{aligned} \tag{3.5}$$

If we imagine to approximate $u$ with a NN, i.e. $u(x,y) \approx u_{NN}(x,y;\boldsymbol{\omega})$, we obtain the following expressions for the residuals:

$$\begin{aligned}
&\mathcal{R}_{pde}(x,y;\boldsymbol{\omega}) = \mathcal{A}[u_{NN}(x,y;\boldsymbol{\omega})](x,y) - \mathcal{F}(x,y), \quad \text{for } (x,y) \in \Omega,\\
&\mathcal{R}_{D}(x,y;\boldsymbol{\omega}) = u_{NN}(x,y;\boldsymbol{\omega}) - g_D(x,y), \quad \text{for } (x,y) \in \Gamma_D,\\
&\mathcal{R}_{N}(x,y;\boldsymbol{\omega}) = u_{NN}(x,y;\boldsymbol{\omega}) - g_N(x,y), \quad \text{for } (x,y) \in \Gamma_N.
\end{aligned} \tag{3.6}$$

If we sample points $\{x_i\}_{i=1}^{N_p} \subset \Omega$, $\{x_i\}_{i=1}^{N_D} \subset \Gamma_D$ and $\{x_i\}_{i=1}^{N_N} \subset \Gamma_N$, then, from (3.6), we can easily compute the loss function of the PINN. More specifically we take the Mean Squared Error $L(r) = \frac{1}{N}\sum_{i=1}^{N} r(\mathbf{x}_i)^2$, where $N$ is the number of points sampled. In this case the mean squared errors are:

$$\begin{aligned}
MSE_p &= L(\mathcal{R}_{pde}(x_i,y_i;\boldsymbol{\omega})) = \frac{1}{N_p}\sum_{i=1}^{N_p}\Big[\mathcal{R}_{pde}(x_i,y_i;\boldsymbol{\omega})\Big]^2,\\
MSE_D &= L(\mathcal{R}_{D}(x_i,y_i;\boldsymbol{\omega})) = \frac{1}{N_D}\sum_{i=1}^{N_D}\Big[\mathcal{R}_{D}(x_i,y_i;\boldsymbol{\omega})\Big]^2,\\
MSE_N &= L(\mathcal{R}_{N}(x_i,y_i;\boldsymbol{\omega})) = \frac{1}{N_N}\sum_{i=1}^{N_N}\Big[\mathcal{R}_{N}(x_i,y_i;\boldsymbol{\omega})\Big]^2.
\end{aligned} \tag{3.7}$$

Globally, the loss function to minimize is

$$L(\boldsymbol{\omega}) = MSE_p + MSE_D + MSE_N. \tag{3.8}$$

The introduced pipeline can be easily generalized to time dependent problems, where also the information about the initial conditions must be integrated, and to the case of systems of partial differential equations with multiple unknowns, which are the ones we are going to deal with.

Now we introduce a generic system of PDEs neglecting, with a slight abuse of formality, the boundary conditions: in fact, in this framework, the methodology of PINN does not change if we deal with boundary conditions or equations, thus for the following system we will only consider the equations, knowing that the methodology can be extended easily to boundary conditions:

$$\begin{cases}
\mathcal{A}[u,v,w](\mathbf{x}) = \mathcal{F}_1(\mathbf{x}), & \text{in } \Omega,\\
\mathcal{B}[u,v,w](\mathbf{x}) = \mathcal{F}_2(\mathbf{x}), & \text{in } \Omega,\\
\mathcal{C}[u,v,w](\mathbf{x}) = \mathcal{F}_3(\mathbf{x}), & \text{in } \Omega,
\end{cases} \tag{3.9}$$

where $u$, $v$, $w$ are the unknowns of the system and $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ are generic differential operators. Then the loss function of (3.9) can be written as

$$
\begin{aligned}
L(\boldsymbol{\omega}) = \frac{1}{N_p}\sum_{i=1}^{N_p}\Big(&\Big[\mathcal{A}[u_{NN}(\mathbf{x}_i), v_{NN}(\mathbf{x}_i), w_{NN}(\mathbf{x}_i)] - \mathcal{F}_1(\mathbf{x}_i)\Big]^2 \\
&+ \Big[\mathcal{B}[u_{NN}(\mathbf{x}_i), v_{NN}(\mathbf{x}_i), w_{NN}(\mathbf{x}_i)] - \mathcal{F}_2(\mathbf{x}_i)\Big]^2 \\
&+ \Big[\mathcal{C}[u_{NN}(\mathbf{x}_i), v_{NN}(\mathbf{x}_i), w_{NN}(\mathbf{x}_i)] - \mathcal{F}_3(\mathbf{x}_i)\Big]^2\Big),
\end{aligned}
\tag{3.10}
$$

which allows us to consider the information given by every condition of (3.9).

## 3.4 Neural Tangent Kernel and Spectral Bias

An important aspect for understanding the learning of PINNs is analyzing the NTK. It is a concept that emerged from the theory of deep NN, offering insights into how networks behave during optimization.
Learning the network parameters $\boldsymbol{\omega}$, implies minimization of the loss function (3.8). Assuming an infinitesimally small learning rate, gradient descent can be expressed using the continuous-time differential equation known as gradient flow, [21], if

$$
\boldsymbol{\omega}_{new} = \boldsymbol{\omega}_{old} - \alpha\nabla_{\omega}L(\boldsymbol{\omega}),
\tag{3.11}
$$

is the optimization step, then $\boldsymbol{\omega}_{new} \to \boldsymbol{\omega}_{old}$ as $\alpha \to 0$, thus

$$
\frac{d\boldsymbol{\omega}}{dt} = -\nabla_{\omega}L(\boldsymbol{\omega}),
\tag{3.12}
$$

if we introduce $t$ as the fictitious time stream. Let us neglect the Neumann term in (3.8) for the sake of simplicity, then, computing the gradient, we obtain:

$$
\begin{aligned}
\frac{d\boldsymbol{\omega}}{dt} = -\Big[&\sum_{i=1}^{N_p}(\mathcal{A}[u(\mathbf{x}_{p,i}, \boldsymbol{\omega}(t))]\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,i}, \boldsymbol{\omega}(t))]}{\partial\boldsymbol{\omega}} \\
&+\sum_{i=1}^{N_D}([u(\mathbf{x}_{D,i}, \boldsymbol{\omega}(t)) - g_D(\mathbf{x}_i)]\frac{\partial[u(\mathbf{x}_{D,i}, \boldsymbol{\omega}(t))]}{\partial\boldsymbol{\omega}}\Big].
\end{aligned}
\tag{3.13}
$$

To be clear with the notation we remind that $\frac{\partial f}{\partial\omega}$ means the derivative of $f$ with respect to every element of $\boldsymbol{\omega}$, all collected in the same vector. From now on we will simplify the notation $\boldsymbol{\omega}(t) \to \boldsymbol{\omega}$.
In general using the chain rule we can state that:

$$
\begin{aligned}
\frac{d\mathcal{A}[u(\mathbf{x}_{p,k}, \boldsymbol{\omega})]}{dt} &= \frac{\mathcal{A}[u(\mathbf{x}_{p,k}, \boldsymbol{\omega})]}{d\boldsymbol{\omega}} \cdot \frac{d\boldsymbol{\omega}}{dt}, \\
\frac{du(\mathbf{x}_{D,j}, \boldsymbol{\omega})}{dt} &= \frac{du(\mathbf{x}_{D,j}, \boldsymbol{\omega})}{d\boldsymbol{\omega}} \cdot \frac{d\boldsymbol{\omega}}{dt}.
\end{aligned}
\tag{3.14}
$$

Inserting equation (3.13) in equation (3.14) we obtain the following expressions

$$\begin{aligned}\frac{d\mathcal{A}[u(\mathbf{x}_{p,k},\boldsymbol{\omega})]}{dt} = -\Big[\sum_{i=1}^{N_p}(\mathcal{A}[u(\mathbf{x}_{p,i},\boldsymbol{\omega})]\ \langle\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,k},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}},\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,i},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}}\rangle \\ + \sum_{i=1}^{N_D}([u(\mathbf{x}_{D,i},\boldsymbol{\omega}) - g_D(\mathbf{x}_i)]\ \langle\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,k},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}},\frac{\partial[u(\mathbf{x}_{D,i},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}}\rangle\Big],\end{aligned} \tag{3.15}$$

and

$$\begin{aligned}\frac{du(\mathbf{x}_{D,j},\boldsymbol{\omega})}{dt} = -\Big[\sum_{i=1}^{N_p}(\mathcal{A}[u(\mathbf{x}_{p,i},\boldsymbol{\omega})]\langle\frac{\partial[u(\mathbf{x}_{D,j},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}},\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,i},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}}\rangle \\ + \sum_{i=1}^{N_D}([u(\mathbf{x}_{D,i},\boldsymbol{\omega}) - g_D(\mathbf{x}_i)]\langle\frac{\partial[u(\mathbf{x}_{D,j},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}},\frac{\partial[u(\mathbf{x}_{D,i},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}}\rangle\Big],\end{aligned} \tag{3.16}$$

where $\langle.\rangle$ represents the $l^2$ scalar product. Let us define the following kernels:

$$\begin{aligned}(\boldsymbol{K}_r)_{j,i}(t) &= \langle\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,j},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}},\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,i},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}}\rangle, \\ (\boldsymbol{K}_b)_{j,i}(t) &= \langle\frac{\partial[u(\mathbf{x}_{D,j},\boldsymbol{\omega})]}{\partial\boldsymbol{\omega}},\frac{\partial[u(\mathbf{x}_{D,i},\boldsymbol{\omega}]}{\partial\boldsymbol{\omega}}\rangle, \\ (\boldsymbol{K}_{rb})_{j,i}(t) &= \langle\frac{\partial[u(\mathbf{x}_{D,j},\boldsymbol{\omega}]}{\partial\boldsymbol{\omega}},\frac{\partial\mathcal{A}[u(\mathbf{x}_{p,i},\boldsymbol{\omega}]}{\partial\boldsymbol{\omega}}\rangle.\end{aligned} \tag{3.17}$$

At this point we can rewrite the equations (3.14) remembering that $\mathbf{x}_p = \{\mathbf{x}_{p,k}\}$ and $\mathbf{x}_D = \{\mathbf{x}_{D,j}\}$:

$$\frac{d}{dt}\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(t))] \\ u(\mathbf{x}_D,\boldsymbol{\omega})\end{pmatrix} = -\boldsymbol{K}(t)\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(t))] \\ u(\mathbf{x}_D,\boldsymbol{\omega}(t)) - g_D(\mathbf{x})\end{pmatrix}, \tag{3.18}$$

where

$$\boldsymbol{K}(t) = \begin{pmatrix}\boldsymbol{K}_b(t) & \boldsymbol{K}_{rb}^T(t) \\ \boldsymbol{K}_{rb}(t) & \boldsymbol{K}_r(t)\end{pmatrix}, \tag{3.19}$$

is the NTK, with $\boldsymbol{K}_b(t) \in \mathbb{R}^{N_D\times N_D}$, $\boldsymbol{K}_r(t) \in \mathbb{R}^{N_p\times N_p}$ and $\boldsymbol{K}_{rb}(t) \in \mathbb{R}^{N_p\times N_D}$. The NTK is often studied in the limit of the width that goes to infinity. One of the most important results from [21] is the following:

**Theorem 7** *Let us take a PINN with one layer and $N$ neurons. If $N \to \infty$, then the NTK defined in* (3.17) *converges in probability to a deterministic kernel $\boldsymbol{K}^*$, which does not depend on the fictitious time $t$.*

In other words we can state that the NTK stays almost constant during all the training phase, thus

$$\boldsymbol{K}^* \approx \boldsymbol{K}(\boldsymbol{t}), \quad \forall t, \tag{3.20}$$

then the system of ordinary differential equation (3.18) can be simplified into

$$\frac{d}{dt}\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega})]\\ u(\mathbf{x}_D,\boldsymbol{\omega})\end{pmatrix} = -\boldsymbol{K}^*\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega})]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(t)) - g_D(\mathbf{x}_D)\end{pmatrix}. \tag{3.21}$$

Since $g_D$ does not depend on $t$ we can rewrite it as

$$\frac{d}{dt}\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega})]\\ u(\mathbf{x}_D,\boldsymbol{\omega}) - g_D(\mathbf{x}_D)\end{pmatrix} = -\boldsymbol{K}^*\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega})]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(t)) - g_D(\mathbf{x}_D)\end{pmatrix}. \tag{3.22}$$

The system (3.22) can be solved easily to obtain

$$\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(t))]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(t)) - g_D(\mathbf{x}_D)\end{pmatrix} = e^{-\boldsymbol{K}^* t}\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(0))]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(0)) - g_D(\mathbf{x}_D)\end{pmatrix}. \tag{3.23}$$

Another way of computing the NTK is shown in [21, Remark 3.3]: if $\boldsymbol{J}_r(t)$ and $\boldsymbol{J}_b(t)$ are the Jacobian matrices of $\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega})]$ and $u(\mathbf{x}_D,\boldsymbol{\omega})$ respectively, then it is easy to see that

$$\boldsymbol{K}(t) = \begin{pmatrix}\boldsymbol{J}_r(t)\\ \boldsymbol{J}_b(t)\end{pmatrix}\begin{pmatrix}\boldsymbol{J}_r^T(t) & \boldsymbol{J}_b^T(t)\end{pmatrix}. \tag{3.24}$$

Of course this result is extended also to the limit kernel $\boldsymbol{K}^*$. Since $\boldsymbol{K}^*$ can be written as the product of a matrix with the transposed itself, then $\boldsymbol{K}^*$ is symmetric and positive definite. From the spectral decomposition of a symmetric matrix we have that:

$$\boldsymbol{K}^* = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^T, \tag{3.25}$$

where $\boldsymbol{P}$ is an orthogonal matrix containing the eigenvectors of the NTK and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues of $\boldsymbol{K}^*$, which are all positive because $\boldsymbol{K}^*$ is positive definite. If we use (3.25) in equation (3.23) we obtain

$$\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(t))]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(t)) - g_D(\mathbf{x}_D)\end{pmatrix} = \boldsymbol{P}e^{-\boldsymbol{\Lambda}t}\boldsymbol{P}^T\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(0))]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(0)) - g_D(\mathbf{x}_D)\end{pmatrix}, \tag{3.26}$$

$$\boldsymbol{P}^T\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(t))]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(t)) - g_D(\mathbf{x}_D)\end{pmatrix} = e^{-\boldsymbol{\Lambda}t}\boldsymbol{P}^T\begin{pmatrix}\mathcal{A}[u(\mathbf{x}_p,\boldsymbol{\omega}(0))]\\ u(\mathbf{x}_D,\boldsymbol{\omega}(0)) - g_D(\mathbf{x}_D)\end{pmatrix}. \tag{3.27}$$

If

$$\boldsymbol{R}(t) = \boldsymbol{P}^T \begin{pmatrix} \mathcal{A}[u(\mathbf{x}_p, \boldsymbol{\omega}(t))] \\ u(\mathbf{x}_D, \boldsymbol{\omega}(t)) - g_D(\mathbf{x}_D) \end{pmatrix} \tag{3.28}$$

is the vector containing the residuals, then equation (3.27) can be written as

$$\boldsymbol{R}(t) = e^{-\boldsymbol{\Lambda} t} \boldsymbol{R}(0). \tag{3.29}$$

Equation (3.29) tells us that the learning of the NN is better explained if we pull the residual into the space generated by the eigenvectors of the NTK. In particular we can say that: as $t \to \infty$ the residuals associated to the PINN go to zero. Since $\boldsymbol{\Lambda}$ is diagonal, the equations (3.29) are decoupled, then the rate of convergence of each pulled residual is related to the respective eigenvalue $\lambda_i$. Let us take two residuals of the equation: $r_i = \mathcal{A}[u(\mathbf{x}_{p,i}, \boldsymbol{\omega})]$ and $r_j = \mathcal{A}[u(\mathbf{x}_{p,j}, \boldsymbol{\omega})]$ with associated eigenvalues $\lambda_i$ and $\lambda_j$ respectively. If $\lambda_i \gg \lambda_j$, then $r_i \to 0$ much faster than $r_j$. This is the Spectral Bias: the PINN learn much faster the pulled residuals with higher eigenvalues and to get accuracy on those with lower ones the training should go on for a very long time. For these reason we say that the PINN is biased towards some residuals than others.

## 3.5 Fourier Feature Embedding

Fourier Feature Embedding (FFE) is a special feature that can be added in the architecture of a NN. It has been introduced in [24], but its implementation in PINNs is also popular [25].
FFE consists in an expansion of the input of the NN with the following map:

$$f(\mathbf{x}) = \begin{bmatrix} \cos(\mathbf{B} \cdot \mathbf{x}) \\ \sin(\mathbf{B} \cdot \mathbf{x}) \end{bmatrix}, \tag{3.30}$$

where $\mathbf{B} \in \mathbb{R}^{m \times d}$, $\mathbf{x} \in \mathbb{R}^d$ is the input of the NN. The parameter $m$ represents the magnitude of the expansion that we want to perform, while the entries of the matrix $b_{i,j} = (\boldsymbol{B})_{i,j}$ are sampled from a Gaussian distribution $b_{i,j} \sim N(0, \tau^2)$, where $\tau$ is a new hyper-parameter.
This Fourier layer is placed right after the input and can be performed with multiple independent embeddings as shown in Figure 3.2.
Let us now take a toy problem with a one-dimensional input $x$ and a network with one FFE and only one layer without bias, namely

$$f(x) = \frac{1}{\sqrt{m}} \boldsymbol{W} \begin{bmatrix} \cos(\mathbf{B} \cdot x) \\ \sin(\mathbf{B} \cdot x) \end{bmatrix} \tag{3.31}$$

where $\boldsymbol{W} \in \mathbb{R}^{1\times 2m}$ are all the trainable weights of the NN. Then, according to the definition (3.17) and taking a set $\{x_i\}_{i=1}^N$ of training points, the NTK is

$$\boldsymbol{K}_{i,j} = \boldsymbol{K}(x_i, x_j) = \frac{1}{m}\begin{bmatrix}\cos(\mathbf{B}\cdot x)\\ \sin(\mathbf{B}\cdot x)\end{bmatrix}^T\begin{bmatrix}\cos(\mathbf{B}\cdot x)\\ \sin(\mathbf{B}\cdot x)\end{bmatrix} = \frac{1}{m}\sum_{k=1}^{m}\cos(b_k(x_i - x_j)). \tag{3.32}$$

In order to better study the eigenvalues we consider the case with $m = 1$, then

$$\boldsymbol{K}(x, y) = \cos(b(x - y)). \tag{3.33}$$

Since (3.33) is a sufficiently easy kernel, from [25, Proposition 3.2] we report the following result:

**Proposition 1** *For the kernel function $\boldsymbol{K}(x, y) = \cos(b(x - y))$, the non-zero eigenvalues are given by*

$$\lambda = \frac{1 \pm \frac{\sin b}{b}}{2}, \tag{3.34}$$

*and the corresponding eigenfunctions are*

$$g(x) = C_1\cos(bx) + C_2\sin(bx). \tag{3.35}$$

Since $b$ is drawn from a Gaussian distribution $N(0, \tau^2)$, a larger choice of $\tau$ increases the likelihood of $b$ having a higher magnitude. Consequently, a large $\sigma$ tends to produce high-frequency eigenfunctions and smaller eigenvalue gaps. This suggests that Fourier features can help mitigate Spectral Bias, allowing for faster convergence to the high-frequency components of a target function.

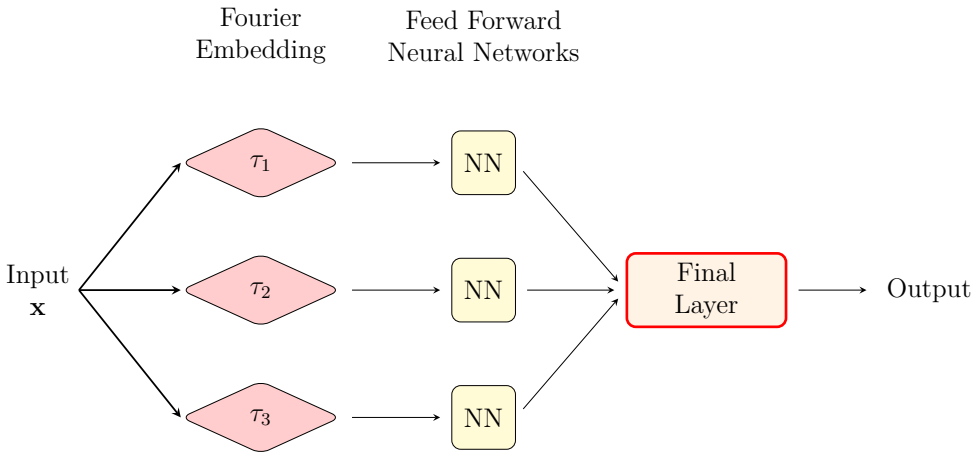



**Figure 3.2:** Neural Network with 3 Fourier embeddings with different $\tau$.

## 3.6 Adaptive weights

In this section we report the Algorithm proposed in [21] which aims to mitigate SB. This approach is very different from FFE, primarily because it does not involve a change in the architecture, but only in the loss function.
In general the loss function of a problem can be written as

$$L(\boldsymbol{\omega}) = L_p(\boldsymbol{\omega}) + L_b(\boldsymbol{\omega}), \tag{3.36}$$

where $L_p$ is the loss related to the residual, while $L_b$ is the loss related to the boundary conditions. One of the main issues of this approach is that the two contributes represent different features of the problem and, in general, they can show great differences in magnitude, thus this could lead the net to learn heterogeneously, i.e. to suffer the SB. In general is desirable to multiply the two contributes for some weights $\beta_p$ and $\beta_b$ respectively

some weights $\beta_p$ and $\beta_b$ to the two contributes, in order to make them reach the same order of magnitude:

$$L(\boldsymbol{\omega}) = \beta_p L_p(\boldsymbol{\omega}) + \beta_b L_b(\boldsymbol{\omega}). \tag{3.37}$$

However, the approach of tuning $\beta_p$ and $\beta_b$ in order to find a good balance is heuristic and not systematic.

Let us recall the system (3.9): in what follows we will not make differences between boundary conditions and equations, since all the machinery can be easily extended for boundary conditions. The NTK of (3.9) can be written as

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{J}_{\mathcal{A}} \\ \boldsymbol{J}_{\mathcal{B}} \\ \boldsymbol{J}_{\mathcal{C}} \end{pmatrix} \begin{pmatrix} \boldsymbol{J}_{\mathcal{A}}^T & \boldsymbol{J}_{\mathcal{B}}^T & \boldsymbol{J}_{\mathcal{C}}^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{K}_{\mathcal{A}} & \boldsymbol{K}_{\mathcal{A},\mathcal{B}} & \boldsymbol{K}_{\mathcal{A},\mathcal{C}} \\ \boldsymbol{K}_{\mathcal{A},\mathcal{B}} & \boldsymbol{K}_{\mathcal{B}} & \boldsymbol{K}_{\mathcal{B},\mathcal{C}} \\ \boldsymbol{K}_{\mathcal{A},\mathcal{C}} & \boldsymbol{K}_{\mathcal{B},\mathcal{C}} & \boldsymbol{K}_{\mathcal{C}} \end{pmatrix}, \tag{3.38}$$

where we exploited (3.19). Suppose we take $N_{\mathcal{A}}, N_{\mathcal{B}}, N_{\mathcal{C}}$ training points for each condition respectively. Then we can write the following:

**Algorithm 1** Adaptive weights

Consider the following loss function

$$L(\boldsymbol{\omega}) = \beta_{\mathcal{A}} L_{\mathcal{A}}(\boldsymbol{\omega}) + \beta_{\mathcal{B}} L_{\mathcal{B}}(\boldsymbol{\omega}) + \beta_{\mathcal{C}} L_{\mathcal{C}}(\boldsymbol{\omega}). \tag{3.39}$$

Update the weights $\boldsymbol{\omega}$ for $S$ epochs in the following way:

**for** $i = 1...$ **do**

Compute $\beta_{\mathcal{A}}, \beta_{\mathcal{B}}, \beta_{\mathcal{C}}$:

$$\begin{aligned} \beta_{\mathcal{A}} &= \frac{\sum_{j=1}^{N_{\mathcal{A}}+N_{\mathcal{B}}+N_{\mathcal{C}}} \lambda_j}{\sum_{j=1}^{N_{\mathcal{A}}} \lambda_j^{\mathcal{A}}} = \frac{\mathrm{Tr}(\boldsymbol{K})}{\mathrm{Tr}(\boldsymbol{K}_{\mathcal{A}})}, \\ \beta_{\mathcal{B}} &= \frac{\sum_{j=1}^{N_{\mathcal{A}}+N_{\mathcal{B}}+N_{\mathcal{C}}} \lambda_j}{\sum_{j=1}^{N_{\mathcal{B}}} \lambda_j^{\mathcal{B}}} = \frac{\mathrm{Tr}(\boldsymbol{K})}{\mathrm{Tr}(\boldsymbol{K}_{\mathcal{B}})}, \\ \beta_{\mathcal{C}} &= \frac{\sum_{j=1}^{N_{\mathcal{A}}+N_{\mathcal{B}}+N_{\mathcal{C}}} \lambda_j}{\sum_{j=1}^{N_{\mathcal{C}}} \lambda_j^{\mathcal{C}}} = \frac{\mathrm{Tr}(\boldsymbol{K})}{\mathrm{Tr}(\boldsymbol{K}_{\mathcal{C}})}. \end{aligned} \tag{3.40}$$

Compute the gradient iteration step using the formula (3.39):

$$\boldsymbol{\omega}_{i+1} = \boldsymbol{\omega}_i - \alpha \nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}_i) \tag{3.41}$$

**end for**

Where $\lambda_j^{\mathcal{A}}, \lambda_j^{\mathcal{B}}, \lambda_j^{\mathcal{C}}$ represent the eigenvalues of $\boldsymbol{K}_{\mathcal{A}}, \boldsymbol{K}_{\mathcal{B}}, \boldsymbol{K}_{\mathcal{C}}$ respectively. The idea of this algorithm is strictly related to NTK theory we developed earlier. First, we have to notice that

$$\begin{aligned} C := \sum_{j=1}^{N_{\mathcal{A}}+N_{\mathcal{B}}+N_{\mathcal{C}}} \lambda_j &= \mathrm{Tr}(\boldsymbol{K}) = \mathrm{Tr}\left[ \begin{pmatrix} \boldsymbol{J}_{\mathcal{A}} \\ \boldsymbol{J}_{\mathcal{B}} \\ \boldsymbol{J}_{\mathcal{C}} \end{pmatrix} \begin{pmatrix} \boldsymbol{J}_{\mathcal{A}}^T & \boldsymbol{J}_{\mathcal{B}}^T & \boldsymbol{J}_{\mathcal{C}}^T \end{pmatrix} \right] \\ &= \mathrm{Tr}(\boldsymbol{J}_{\mathcal{A}}\boldsymbol{J}_{\mathcal{A}}^T + \boldsymbol{J}_{\mathcal{B}}\boldsymbol{J}_{\mathcal{B}}^T + \boldsymbol{J}_{\mathcal{C}}\boldsymbol{J}_{\mathcal{C}}^T) \\ &= \mathrm{Tr}(\boldsymbol{J}_{\mathcal{A}}\boldsymbol{J}_{\mathcal{A}}^T) + \mathrm{Tr}(\boldsymbol{J}_{\mathcal{B}}\boldsymbol{J}_{\mathcal{B}}^T) + \mathrm{Tr}(\boldsymbol{J}_{\mathcal{C}}\boldsymbol{J}_{\mathcal{C}}^T) \\ &= \sum_{j=1}^{N_{\mathcal{A}}} \lambda_j^{\mathcal{A}} + \sum_{j=1}^{N_{\mathcal{B}}} \lambda_j^{\mathcal{B}} + \sum_{j=1}^{N_{\mathcal{C}}} \lambda_j^{\mathcal{C}} \\ &= C_{\mathcal{A}} + C_{\mathcal{B}} + C_{\mathcal{C}}. \end{aligned} \tag{3.42}$$

If, for instance, $C_{\mathcal{A}} \ll C_{\mathcal{B}}$, then from NTK theory we know that the NN will learn faster the condition described by the differential operator $\mathcal{B}$ than the condition described by the operator $\mathcal{A}$. Thus in general the rate of convergence of the terms can vary greatly. The weights $\beta_{\mathcal{A}}, \beta_{\mathcal{B}}$ and $\beta_{\mathcal{C}}$ can be written as

$$\begin{aligned}\beta_{\mathcal{A}} &= \frac{C}{C_{\mathcal{A}}} = \frac{C_{\mathcal{A}} + C_{\mathcal{B}} + C_{\mathcal{C}}}{C_{\mathcal{A}}},\\ \beta_{\mathcal{B}} &= \frac{C}{C_{\mathcal{B}}} = \frac{C_{\mathcal{A}} + C_{\mathcal{B}} + C_{\mathcal{C}}}{C_{\mathcal{B}}},\\ \beta_{\mathcal{C}} &= \frac{C}{C_{\mathcal{C}}} = \frac{C_{\mathcal{A}} + C_{\mathcal{B}} + C_{\mathcal{C}}}{C_{\mathcal{C}}},\end{aligned} \tag{3.43}$$

thus, if $C_{\mathcal{A}} \approx C_{\mathcal{C}} \ll C_{\mathcal{B}}$, then

$$\begin{aligned}\beta_{\mathcal{A}} &\approx \frac{C_{\mathcal{B}}}{C_{\mathcal{A}}} \gg 1,\\ \beta_{\mathcal{B}} &\approx \frac{C_{\mathcal{B}}}{C_{\mathcal{B}}} = 1,\\ \beta_{\mathcal{C}} &\approx \frac{C_{\mathcal{B}}}{C_{\mathcal{C}}} \gg 1,\end{aligned} \tag{3.44}$$

which means that, the condition $\mathcal{B}$ will be learned maintaining the same speed of convergence, while the conditions $\mathcal{A}$ and $\mathcal{C}$ will be learned faster. This is a good approach for leveling the speed of convergence of all the conditions and make the learning homogeneous.

Compared to FFE, Algorithm 1 offers the advantage of not modifying the architecture of the PINN and avoids the introduction of any new hyper-parameter requiring careful tuning. On the other hand, Algorithm 1 does not take into account the variations of speed of convergence of the residuals within the same condition, meaning that SB can still create problems in some cases. Additionally, Algorithm 1 requires computation of the NTK, which may be time consuming. Consequently, in the upcoming experiment concerning OCPs, we will compute traces only in the initial step and maintain these weights throughout the training. This approach can be particularly reasonable when training an over-parameterized network.

## 3.7 Implementation aspects

Before discussing the experiments, we explain all the features that we consider relevant for training:

- Strong imposition of boundary conditions.
- Sampling of the physical and the parametric domain.
- An additional input of the problem called "Extra feature".

For what concerns the **strong boundary conditions**, it is well known in literature, see for instance [33], that is possible to cut the mean squared error $MSE_D$ associated to the Dirichlet boundary conditions, and ensuring them by multiplying the output of the NN with an ad-hoc function. For instance, if the boundary conditions are homogeneous like in Equation (5.9), then we could multiply the output of the NN $u_{NN}(x, y; \boldsymbol{\omega})$ with any function that nullifies at the boundary $b(x, y)$ such that $b|_{\partial\Omega} = 0$. Then, the real output of the net becomes the function $u_{NN}(x, y; \boldsymbol{\omega})b(x, y)$ which is clearly zero on the boundary. This procedure usually leads to an improvement of the approximation of the PINN, although the choice of the function $b(x, y)$ is crucial: sometimes it could lead the NN to reach a suboptimal approximation. It is important to recall that this procedure is straightforward to implement for cartesian boundaries, such the ones we are going to deal with, but can be a challenge for more complex boundaries, limiting the possibility of generalization of this technique.

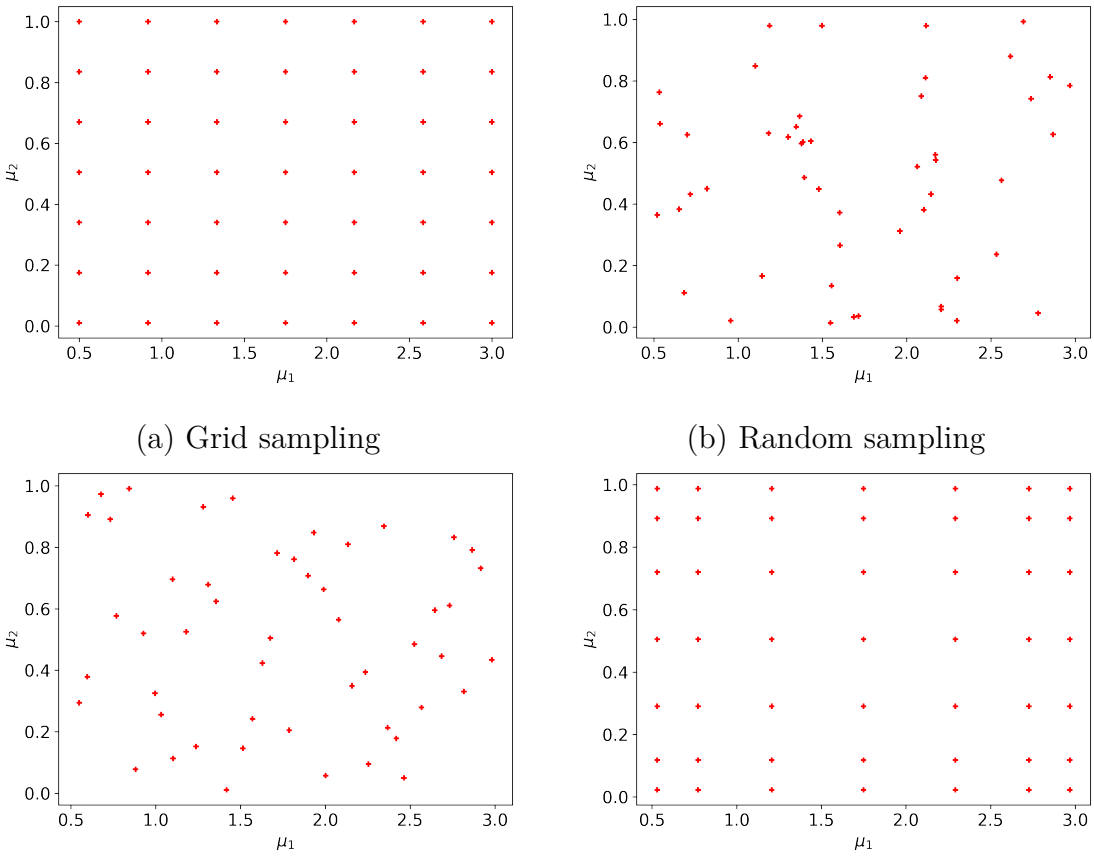
The sampling techniques refer to the methods used to take points both in the physical and the parametric domain. The baseline simulations made for the following problems are made by a random sampling, but we tried others sampling strategies such as "grid" sampling and "Chebyshev" sampling. The "grid" sampling consists in taking equally spaced points on every dimension and then making a cartesian product between them, which is similar to the "Chebyshev" sampling, where the difference is that the second provides more points when is closer to the edges. Examples of sampling strategies, that are also the ones used for the experiments, are provided in Figures 3.3.

The "extra feature" refers to the work [26] in which the authors enforced the physical information provided by the PDE with an additional one. Let us take a function $E(\boldsymbol{x})$ which depends on the other input of the NN, namely $\boldsymbol{x}$. This operation is particularly useful when we want to reach higher level of PINN performance without using a deeper NN, which can lead to the problem of vanishing gradients. The idea is that we can give some information about the physics of the problem to the NN such as the shape of the function or maybe the behaviour of the solution with respect to time. The addition of an extra feature is highly problem dependent, since it should be tuned to get the best performance possible.

Another important enhancement provided by [26] concerns the Physics-Informed Architecture (PIARCH) illustrated in Figure 3.4. This is an innovative way of dealing with OCP with PINNs. In fact, when building the system of differential equation with the Lagrange optimality condition (2.7), one of the equation can always be written in the form

$$z(\mathbf{x}) = \alpha u(\mathbf{x}), \tag{3.45}$$

where $\alpha$ is the penalization term. In the PIARCH framework, equation (3.45) is not considered in the loss function and the network output is restricted excluding $z(\boldsymbol{x})$. The latter is rebuild by strongly imposing the excluded equation as a sequential output of the PINN. The innovation of this feature stands in the fact that we give the right physical information to the NN without the need of having more output and more calculations. It has the same principle of the strong boundary conditions: where we know something about the physics of the problem, there is no need of training the net. In the chapter 5 we will show in detail what the equation included in the PIARCH "step" is.




(a) Grid sampling

(b) Random sampling

(c) Latin Hypercube sampling

(d) Chebyshev sampling

**Figure 3.3:** These plots show the four sampling techniques in the parametric domain $\mathcal{P} = [0.5, 3] \times [0,1]$.

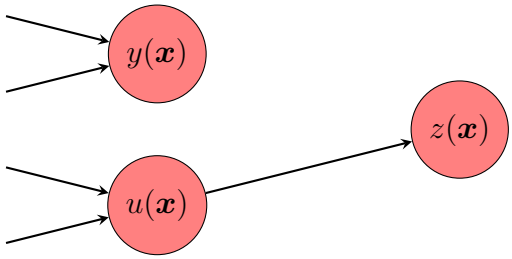



**Figure 3.4:** PIARCH structure: $z(\boldsymbol{x})$ is not a direct output of the NN but it is obtained afterwards.

# Chapter 4

# Practical insights on Fourier Embeddings and Adaptive weights

In this section we deal with some experiments about the FFE in order to show its potentialities and possible limitations. First we show a simple example where FFE overcomes the problem of SB. Then, we show how the FFE behaves with respect to the parameter $\tau$ both for a problem with double frequency and a problem with increasing frequency.

## 4.1 Mitigating Spectral Bias with Fourier Embeddings

Let us take the following problem

$$\begin{cases} \Delta u = \sin(x) + 0.1\alpha^2 \sin(\alpha x) \quad \text{in} \quad (-\pi, \pi), \\ u = 0 \quad \text{for} \quad x = \{-\pi, \pi\}, \end{cases} \tag{4.1}$$

whose solution, $u(x) = \sin(x) + 0.1 \sin(\alpha x)$ depends on the parameter $\alpha$. It is of interest studying how a standard PINN can learn this solution, against a similar architecture that embodies a double Fourier Embedding.
For the standard PINN we took a network with 2 layers with 100 neurons each. We took Softplus as activation function and trained for 8000 epochs with a learning rate of 0.0001.
Concerning the FFE network, addressing to equation (3.30), we took $m = 100$ and

a double embedding with $\tau = 1$ and $\tau = 10$. The two separate embeddings are then passed through a Feed Forward NN with one layer of 100 neurons and then connected with a final linear layer that reunites the output from 200 to one. Again we trained for 8000 epochs and a learning rate equal to 0.0001.
For both systems we took 900 equally spaced points in the domain and only 1 point for each boundary condition. We then trained the network for different values of $\alpha$, expecting that the standard PINN would fail the task as $\alpha$ increases, while FFE PINN should be able to get more accurate results.

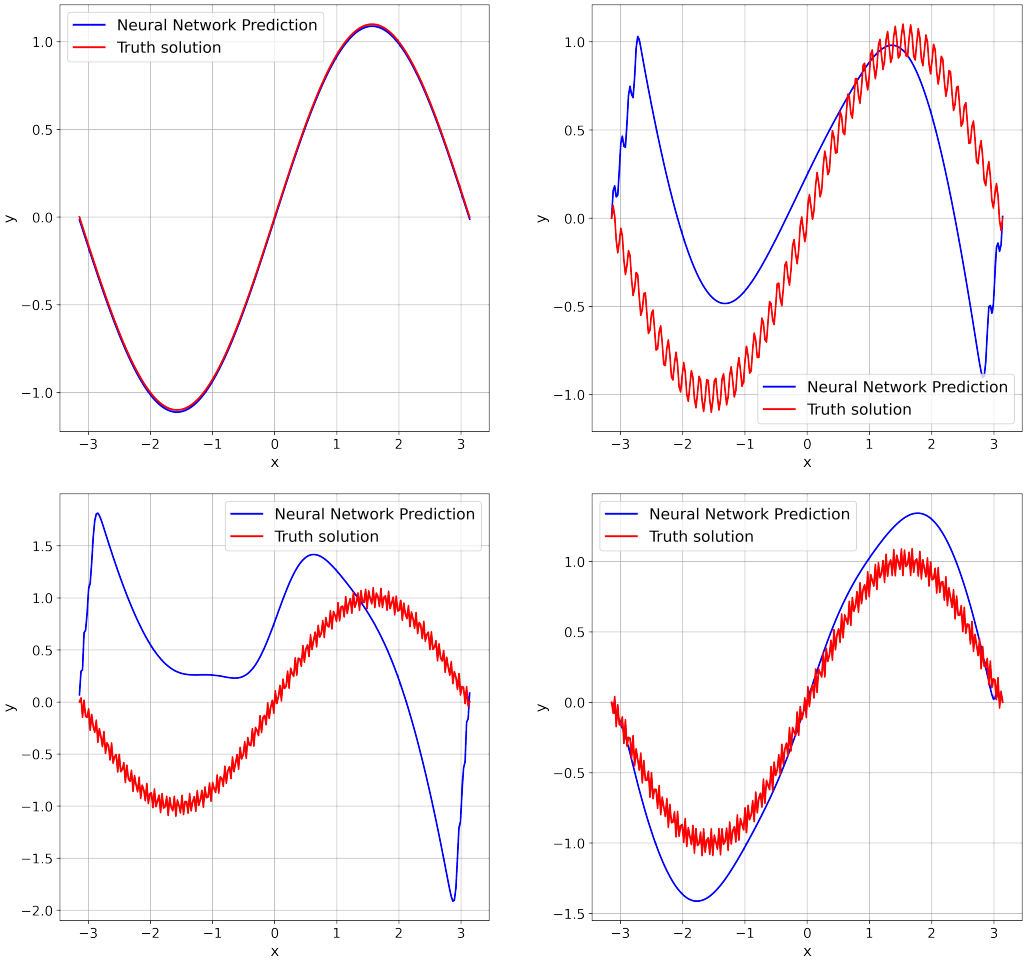



**Figure 4.1:** From top left to bottom right, we have the comparison between the standard PINN and the truth solution of equation (4.1) for $\alpha = 1,50,100,150$, respectively.

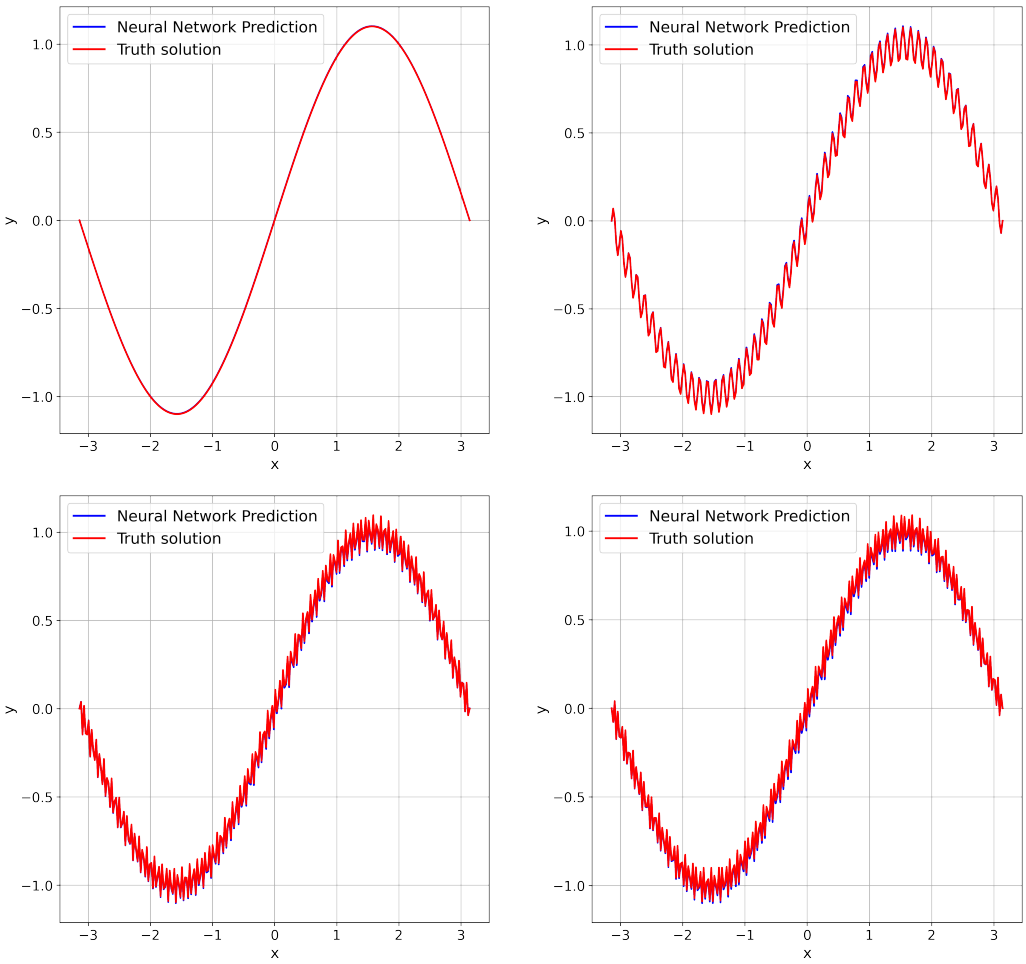



**Figure 4.2:** From top left to bottom right we have the comparison between the PINN enhanced with FFE and the truth solution of equation (4.1) for $\alpha = 1,50,100,150$ respectively.

As we can see from Figure 4.2, FFE is able to reconstruct a good approximation, while standard PINN approach fails already for $\alpha = 50$. It is also interesting to note that FFE PINN is able to deliver a good approximation for all the $\alpha$ that we tried, meaning that FFE is a very robust tool.

## 4.2 Sensitivity on the parameter $\tau$

Always referring to the equation (4.1) we studied how the approximation ability of the FFE changes with respect to the parameter $\tau$. We took a NN with a feature expansion of $m = 100$, only one embedding. After the embedding the architecture includes a Feed Forward NN with one layer and 100 neurons. We trained on 900 training points in the domain for 3000 epochs and a learning rate of 0.0001. We fixed a high frequency, $\alpha = 150$, and we made $\tau$ vary between 0.01 and 10.
Results are shown in Figure 4.3. We can see that, as $\tau$ increases, the PINN is able to identify high variations in the solution, while lower values are more associated to lower frequencies. What it is really interesting to see is that for $\tau = 5$ the PINN almost reconstruct correctly the solution. This behavior is well explained because $\tau = 5$ is a sufficiently big that lets the sampling wander between low values and high values. In fact $\tau = 5$ is in the middle of the values we used, referring to Figure 4.2, that let us recover the correct solution.
These results are aligned to the theoretical results showed in the previous chapter.

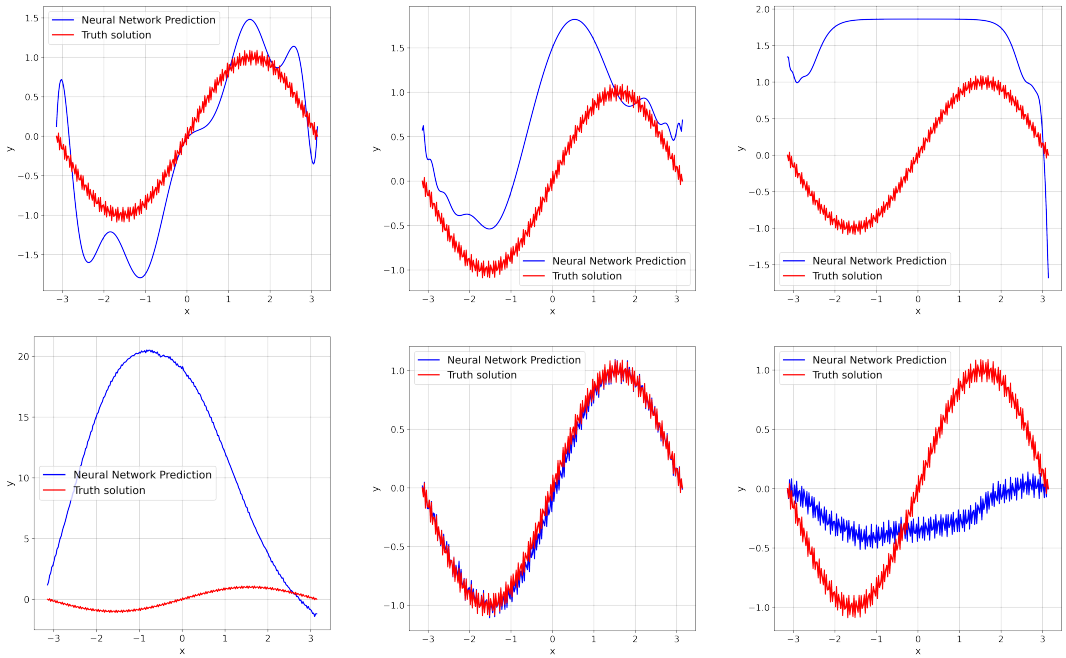



**Figure 4.3:** From top left to bottom right there is the comparison between the truth solution and the PINN approximation with one FFE for $\tau = 0.01,0.05,0.1,0.5,1,5,10$ respectively.

## 4.3 Mitigating Spectral Bias with Adaptive weights

Let us give an example of a one dimensional problem where the Adaptive weights can fasten the convergence towards the truth solution.
Let us consider the following problem:

$$\begin{cases} \Delta u = -16\pi^2 \sin(4\pi x) \quad \text{in} \quad [0,1] \\ u = 0 \quad \text{on} \quad x = 0,1 \end{cases} \tag{4.2}$$

whose solution is $u(x) = \sin(4\pi x)$. We train a PINN with the following hyper-parameter

- $N_b = 50$ on every edge, $N_p = 100$ points in $(0,1)$,
- one layer with 512 neurons,
- activation function: $\sigma(x) = \tanh(x)$,
- learning rate: 0.00001,
- epochs: 40000.

This structure is not sufficient in order to get a good approximation. However if we enhance the training using the Algorithm 1, it is evident, from Figure 4.4, that this approach can make the training much faster using the same hyper-parameters.

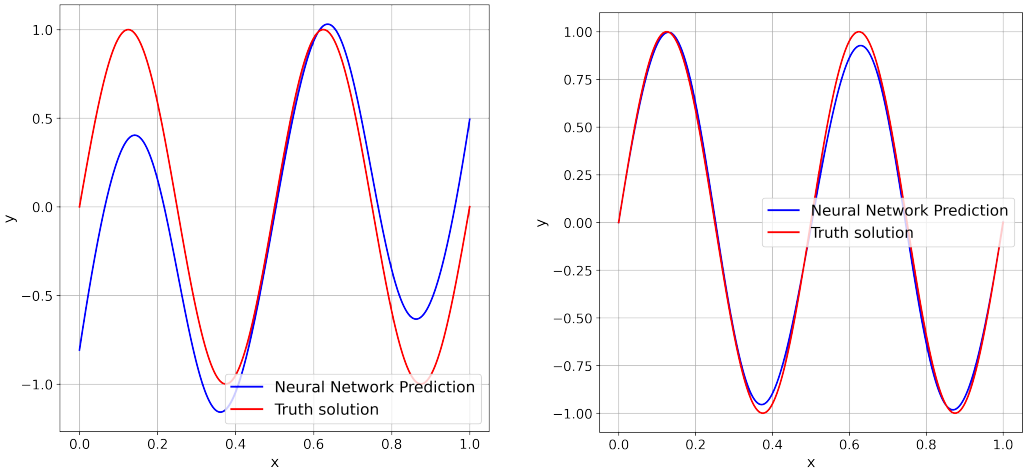



**Figure 4.4:** On the left there is the solution with a standard PINN. On the right the same PINN with the Adaptive weights used as in the Algorithm 1.

In particular, it is evident that the adaptive weights help the PINN to reach the correct boundary conditions. In this case, referring to Algorithm 1, the weight multiplied to the mean squared error of the boundary conditions is much higher than the one multiplied to the mean squared error of the equation. This happens because the SB bring the PINN to learn much faster the equation than the boundary condition.

# Chapter 5

# PINNs for Optimal Control Problems

In this chapter we study two Parametric OCPs. In particular we start from the work [26] and improve the performance of the PINN, studying the relationship between the performance and some hyper-parameters and features of the PINN.

## 5.1 Parametric Elliptic Optimal Control

The first optimal control problem we take into account is a parametric elliptic problem described by the following equations:

$$\min_{y(\mathbf{x},\boldsymbol{\mu}),u(\mathbf{x},\boldsymbol{\mu})} J(y,u) = \frac{1}{2}||y(\mathbf{x},\boldsymbol{\mu}) - \mu_1||^2_{L^2(\Omega)} + \frac{\mu_2}{2}||u(\mathbf{x},\boldsymbol{\mu})||^2_{L^2(\Omega)} \tag{5.1}$$

constrained to

$$\begin{cases} -\Delta y(\mathbf{x},\boldsymbol{\mu}) = u(\mathbf{x},\boldsymbol{\mu}), & \text{in } \Omega \\ y(\mathbf{x},\boldsymbol{\mu}) = 0, & \text{on } \partial\Omega, \end{cases} \tag{5.2}$$

where $\mathbf{x} \in [-1,1] \times [-1,1]$ and $\boldsymbol{\mu} = (\mu_1, \mu_2) \in [0.5,3] \times [0.01,1]$.

Due to our solver based on PINN, we have to formulate the problem above in a more convenient way. We apply the Lagrange theorem in order to get a system of equation that we can solve with our framework.
Now, we proceed to write the Lagrangian system of the parametric elliptic OCP, since we are going to solve that system with a PINN. First, we write the state equation in weak form:

$$a(y, v) = F(u, v), \quad \forall v \in Y, \tag{5.3}$$

where

$$\begin{aligned} a(y, v) &= \int_\Omega \nabla y \cdot \nabla v \, d\Omega, \\ F(u, v) &= \int_\Omega f v \, d\Omega + \int_\Omega uv \, d\Omega. \end{aligned} \tag{5.4}$$

Then we have to write the Lagrangian of the problem, that we recall being the sum of the cost functional with the constraint: $J(y, u) + \langle p, \mathcal{E}(y, u)\rangle$:

$$\begin{aligned} \mathcal{L}(y, u, p) = \frac{1}{2}\int_\Omega (y - \mu_1)^2 \, d\Omega + \frac{\mu_2}{2}\int_\Omega u^2 \, d\Omega + \\ \int_\Omega \nabla y \cdot \nabla p \, d\Omega - \int_\Omega fp \, d\Omega - \int_\Omega up \, d\Omega. \end{aligned} \tag{5.5}$$

Then we can differentiate to get the equations:

$$\begin{aligned} &\langle D_y\mathcal{L}(y, u, z), v\rangle = \int_\Omega (y - \mu_1)v \, d\Omega + \int_\Omega \nabla v \cdot \nabla z \, d\Omega = 0 \quad \forall v \in Y, \\ &\langle D_u\mathcal{L}(y, u, z), \omega\rangle = \mu_2 \int_\Omega u\omega \, d\Omega - \int_\Omega z\omega \, d\Omega = 0 \quad \forall \omega \in U, \\ &\langle D_z\mathcal{L}(y, u, z), q\rangle = \int_\Omega \nabla y \cdot \nabla q \, d\Omega - \int_\Omega fq \, d\Omega - \int_\Omega uq \, d\Omega = 0 \quad \forall q \in Q. \end{aligned} \tag{5.6}$$

Applying the Green's theorem to the first and the third equations and remembering that $v|_{\partial\Omega} = q|_{\partial\Omega} = 0$ we get

$$\begin{aligned} &\int_\Omega \Big((y - \mu_1) + \Delta z\Big)v \, d\Omega = 0 \quad \forall v \in Y, \\ &\int_\Omega \Big(u\mu_2 - z\Big)\omega \, d\Omega = 0 \quad \forall \omega \in U, \\ &\int_\Omega \Big(\Delta y - f - u\Big)q \, d\Omega = 0 \quad \forall q \in Q. \end{aligned} \tag{5.7}$$

Since the variables $v$, $\omega$, $q$ are arbitrary, and taking $f \equiv 0$, then the previous equations can be written as

$$\begin{cases} y(\mathbf{x}, \mu_1, \mu_2) - \Delta z(\mathbf{x}, \mu_1, \mu_2) = \mu_1 \quad \text{in} \quad \Omega, \\ \mu_2 u(\mathbf{x}, \mu_1, \mu_2) = z(\mathbf{x}, \mu_1, \mu_2) \quad \text{in} \quad \Omega, \\ -\Delta y(\mathbf{x}, \mu_1, \mu_2) = u(\mathbf{x}, \mu_1, \mu_2) \quad \text{in} \quad \Omega, \end{cases} \tag{5.8}$$

with the boundary conditions

$$\begin{cases} z(\mathbf{x}, \mu_1, \mu_2) = 0 \quad \text{on} \quad \partial\Omega, \\ y(\mathbf{x}, \mu_1, \mu_2) = 0 \quad \text{on} \quad \partial\Omega. \end{cases} \tag{5.9}$$

The second equation of (5.8) is the PIARCH "step".

For this problem, eight different settings were employed, as shown in Tables 5.1. The baseline simulation was made with the following hyper-parameters:

- Learning rate: 0.0005,
- Neurons in each hidden layer: [40,40,20],
- Optimizer: ADAM,
- Activation function: Softplus,
- Extra feature: $K(\mathbf{x}) = (1 - x_0^2)(1 - x_1^2)$,
- Epochs of training: 50000,
- Sampling: $N_p = 900$ points randomly sampled within $\Omega$, $N_b = 200$ points randomly sampled within $\partial\Omega$ and $N_{\boldsymbol{\mu}} = 50$ points randomly sampled in the parametric domain both for $\Omega$ and $\partial\Omega$.

Every other setting is obtained by modifying one or more of the characteristics above. For example, “Grid-Latin” indicates that grid sampling was used for the physical domain, while Latin hypercube sampling was applied to the parametric domain. In contrast, “Grid” sampling means only the sampling method for the physical domain was changed to grid. Similarly, “Strong BC” or simply “Strong” indicates that strong boundary conditions were enforced. We computed the relative error for all settings and for three points in the parametric domain: $\boldsymbol{\mu} = (3,0.01)$, $\boldsymbol{\mu} = (3,0.1)$, $\boldsymbol{\mu} = (3,1)$.

Control variable $u$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu_2 = 0.01$ | $\mu_2 = 0.1$ | $\mu_2 = 1$ | $\mu_2 = 0.01$ | $\mu_2 = 0.1$ | $\mu_2 = 1$ |
| Baseline | 0.36 | 0.09 | 0.02 | 0.19 | 0.03 | 0.008 |
| No extra feature | 0.58 | 0.17 | 0.09 | 0.49 | 0.04 | 0.023 |
| Grid sampling | **0.28** | 0.22 | 0.01 | 0.15 | 0.08 | 0.008 |
| Strong BC | 0.37 | **0.04** | 0.02 | 0.20 | **0.01** | 0.004 |
| Strong + Grid-Random | 0.40 | 0.06 | **0.006** | 0.19 | 0.05 | 0.003 |
| Strong + Grid-Grid | 0.35 | 0.24 | 0.01 | **0.01** | 0.2 | **0.001** |
| Strong + Grid-Latin | 0.42 | 0.06 | 0.03 | 0.53 | 0.35 | 0.36 |
| Strong + Grid-Chebyshev | 0.36 | 0.06 | 0.007 | 0.18 | 0.03 | 0.002 |

State variable $y$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu_2 = 0.01$ | $\mu_2 = 0.1$ | $\mu_2 = 1$ | $\mu_2 = 0.01$ | $\mu_2 = 0.1$ | $\mu_2 = 1$ |
| Baseline | 0.20 | 0.04 | 0.01 | 0.05 | 0.03 | 0.023 |
| No extra feature | 0.51 | 0.05 | 0.05 | 0.10 | 0.04 | 0.027 |
| Grid sampling | 0.08 | 0.16 | 0.007 | 0.04 | 0.04 | 0.006 |
| Strong BC | 0.19 | **0.01** | 0.02 | 0.05 | 0.02 | 0.02 |
| Strong + Grid-Random | 0.22 | **0.01** | 0.06 | 0.04 | 0.08 | 0.003 |
| Strong + Grid-Grid | **0.07** | 0.24 | **0.003** | **0.003** | 0.13 | 0.005 |
| Strong + Grid-Latin | 0.28 | 0.03 | 0.03 | 0.38 | 0.3 | 0.31 |
| Strong + Grid-Chebyshev | 0.15 | 0.05 | 0.005 | 0.09 | **0.007** | **0.001** |

Adjoint variable $z$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu_2 = 0.01$ | $\mu_2 = 0.1$ | $\mu_2 = 1$ | $\mu_2 = 0.01$ | $\mu_2 = 0.1$ | $\mu_2 = 1$ |
| Baseline | 0.56 | 0.02 | 0.009 | 0.19 | 0.03 | 0.008 |
| No extra feature | 1.48 | 0.04 | 0.03 | 0.49 | 0.04 | 0.023 |
| Grid sampling | 0.17 | 0.03 | 0.006 | 0.15 | 0.08 | 0.008 |
| Strong BC | 0.69 | **0.001** | 0.006 | 0.20 | **0.01** | 0.004 |
| Strong + Grid-Random | 0.7 | 0.008 | 0.01 | 0.19 | 0.05 | 0.003 |
| Strong + Grid-Grid | **0.07** | 0.07 | 0.001 | **0.01** | 0.2 | **0.001** |
| Strong + Grid-Latin | 1.06 | 0.01 | 0.002 | 0.53 | 0.35 | 0.36 |
| Strong + Grid-Chebyshev | 0.44 | 0.01 | **0.001** | 0.18 | 0.03 | 0.002 |

**Table 5.1:** Relative error of the Parametric Elliptic OCP for different settings. The bold text refers to the lowest error between the ones in the column. The tests were made for a fixed $\mu_1 = 3$.

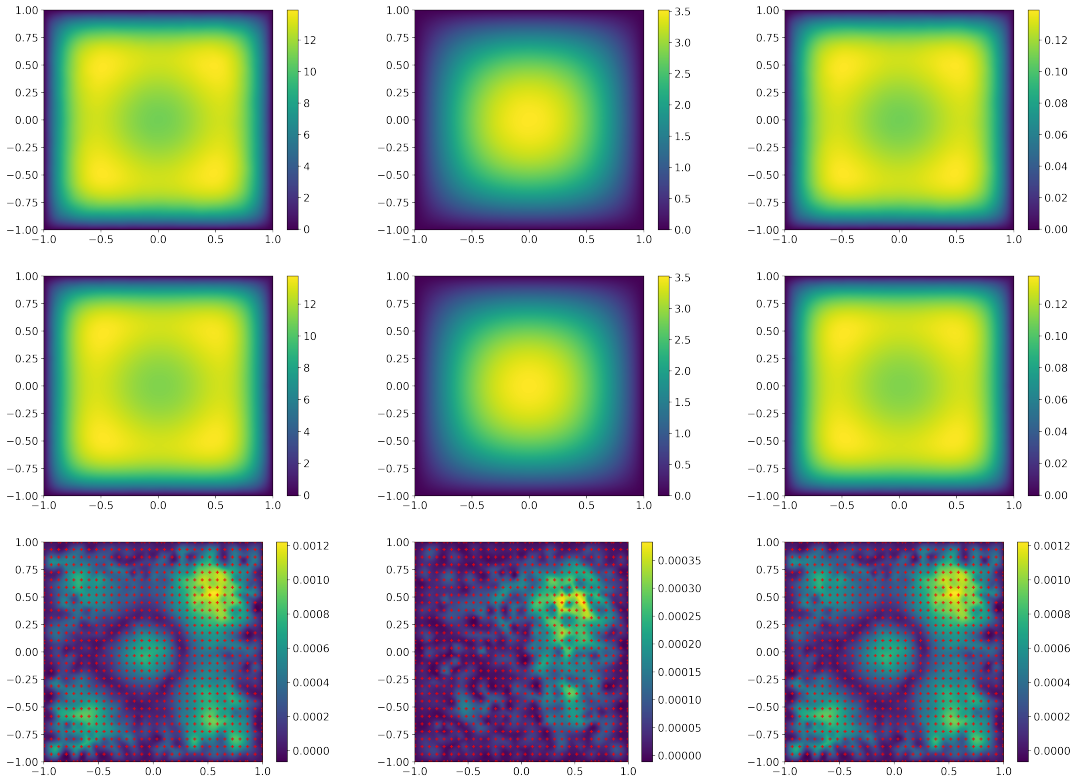


**Figure 5.1:** The finite elements solutions (top row) are shown along with the PIARCH solutions (second row) and the relative error between them (third row) for $\mu_1 = 3$ and $\mu_2 = 0.01$. From left to right the variables are $u$, $y$ and $z$. The red dots represent the training points.

From tables 5.1, it is clear that enforcing strong BC is the most important feature among those that improve the baseline simulation. In fact, the lowest errors were obtained regardless all the other features and couples of $\boldsymbol{\mu}$. Observing the first four settings, where we changed one feature per try, we found out that the extra feature is actually improving the performance of the PINN as expected. Additionally, grid sampling on the physical domain usually performs better except when $\mu_2 = 0.1$ and strong BC performs at least with the same order of magnitude. We also noticed that, generally, PIARCH architecture lowers the error especially on adjoint variable $z$ as expected.

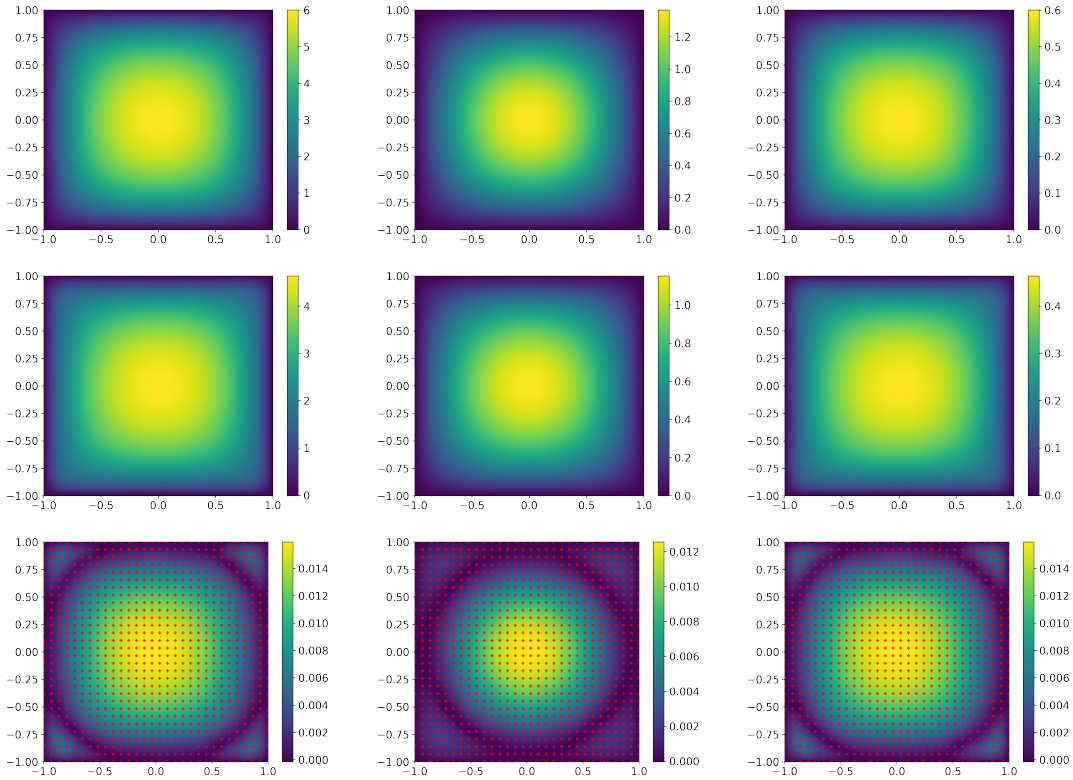


**Figure 5.2:** The finite elements solutions (top row) are shown along with the PIARCH solutions (second row) and the relative error between them (third row) for $\mu_1 = 3$ and $\mu_2 = 0.1$. From left to right the variables are $u$, $y$ and $z$. The red dots represent the training points.

As it can be seen, for the lowest values of $\mu_2$, the error is increasing for almost all settings This is probably due to the fact that when $\mu_2 \ll 1$, i.e. the control is less penalized, the solutions of the problem vary more in the domain in terms of scales, and probably PINN and PIARCH have more difficulties predicting the correct solutions. Mainly for this reason, we tried the setting with Chebyshev sampling in the parametric domain, assuming that samples closer to the edges would help recognize better the less penalized solutions. Promising results were obtained, although sampling more points would probably help.

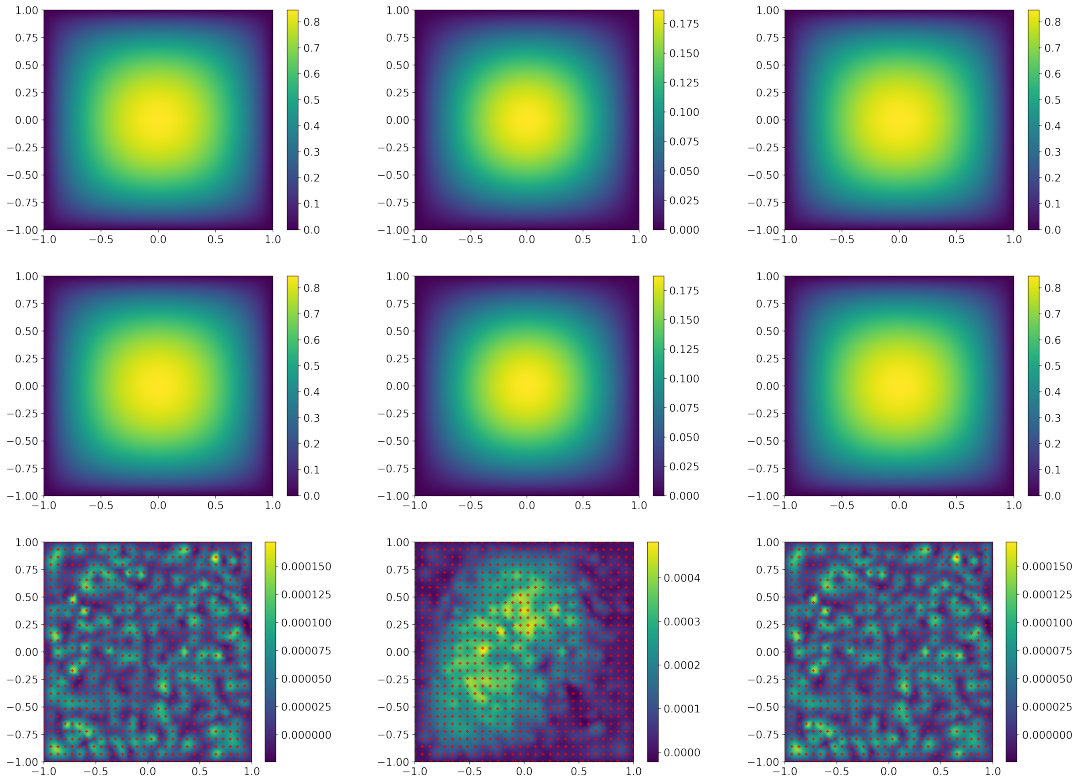


**Figure 5.3:** The finite elements solutions (top row) are shown along with the PIARCH solutions (second row) and the relative error between them (third row) for $\mu_1 = 3$ and $\mu_2 = 1$. From left to right the variables are $u$, $y$ and $z$. The red dots represent the training points.

It should be noted that for the first four settings where random sampling was used everywhere, the error on all the variables is much lower when $\mu_2 = 0.1$. We explain this behaviour by hypothesizing that, although grid sampling usually gives better results, random sampling may outperform it when sufficiently "lucky", i.e. when it samples more points nearby the tested parameters. Sampling more points would likely reduce the probability of having great discrepancies in error between the sampling methods.

Overall, the best approximations were obtained with PIARCH framework, enforcing the strong BC. It is not clear which sampling is the best one, but longer trainings and more sampling in the parametric domain could easily point it out. However, results in Figures 5.1,5.2 and 5.3 are a good agreement in terms of accuracy.

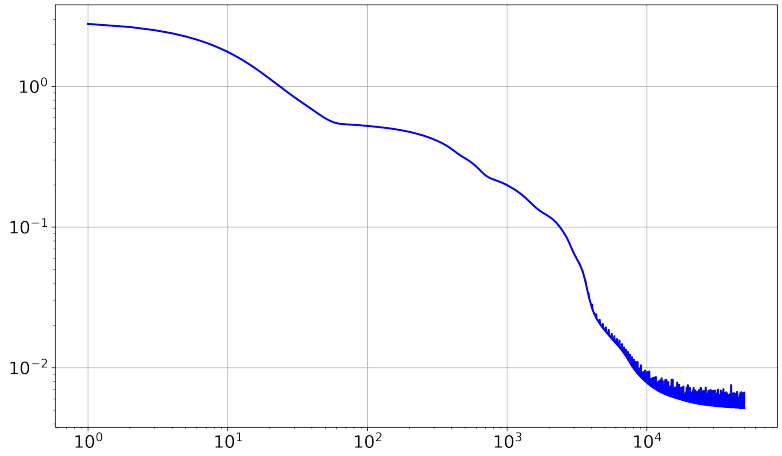



**Figure 5.4:** Trend of the mean of the loss functions of every equation for the Parametric Elliptic OCP.

## 5.2 Parametric Stokes Optimal Control

The second optimal control problem we take into account is a parametric Stokes problem described by the following equations

$$\min_{\mathbf{v}(\mathbf{x},\mu),u(\mathbf{x},\mu)} J(y,u) = ||\mathbf{v}(\mathbf{x},\mu) - x_2||^2_{L^2(\Omega)} + \frac{\alpha}{2}||\mathbf{u}(\mathbf{x},\mu)||^2_{L^2(\Omega)} \tag{5.10}$$

constrained to

$$\begin{cases} -0.1\Delta\mathbf{v}(\mathbf{x};\mu) + \nabla p(\mathbf{x};\mu) = \boldsymbol{f}(\mathbf{x};\mu) + \mathbf{u}(\mathbf{x};\mu) \quad \text{in} \quad \Omega, \\ \nabla \cdot \mathbf{v}(\mathbf{x};\mu) = 0 \quad \text{in} \quad \Omega, \\ v_1(\mathbf{x};\mu) = x_2 \quad \text{and} \quad v_2(\mathbf{x};\mu) = 0 \quad \text{on} \quad \partial\Omega, \\ -p(\mathbf{x};\mu) + 0.1\dfrac{\partial v_1(\mathbf{x};\mu)}{\partial n} = 0 \quad \text{and} \quad v_2(\mathbf{x};\mu) = 0 \quad \text{on} \quad \partial\Omega, \end{cases} \tag{5.11}$$

where the physical domain is $\mathbf{x} \in [0,1] \times [0,2]$ and the parameter domain is $\mu \in [0.5,1.5]$. The constant $\alpha$ is fixed at 0.008 and $f(\mathbf{x},\mu) = [0,\mu]^T$. The main difference from the elliptic problem is that the state variables are three, $v_1$,$v_2$, $p$ and the same is true for the adjoint variables $z_1$, $z_2$, $z_3$. With the same notation used until now, we have that $\mathbf{v} \in Y^2$ and $p \in Q$, where $Y^2$ is the extension of $Y$ into the space of two-dimensional functions. They both represent the state variables, velocity and pressure respectively. As for the problem of parametric elliptic OCP, we need to reformulate the problem in a Lagrangian formulation. First, we write the problem in its weak form:

$$\begin{cases} a(\mathbf{v},\mathbf{w}) + b(\mathbf{w},p) = F(\mathbf{u},\mathbf{w};\mu) \quad \forall \mathbf{w} \in Y^2, \\ b(\mathbf{v},q) = 0 \quad \forall \mathbf{w} \in Q, \end{cases} \tag{5.12}$$

where

$$\begin{aligned} a(\mathbf{v},\mathbf{w}) &= 0.1 \int_\Omega \nabla \mathbf{v} : \nabla \mathbf{w} \, d\Omega, \\ b(\mathbf{v},q) &= -\int_\Omega p \nabla \cdot \mathbf{w} \, d\Omega, \\ F(\mathbf{u},\mathbf{w}) &= \int_\Omega \boldsymbol{f}(\mathbf{x},\mu) \cdot \mathbf{w} \, d\Omega + \int_\Omega \mathbf{u} \cdot \mathbf{w} \, d\Omega. \end{aligned} \tag{5.13}$$

The Lagrangian functional associated to this problem can be written as:

$$\begin{aligned} \mathcal{L}(\mathbf{v},\mathbf{z},\mathbf{u},p,r) =& \frac{1}{2}\int_\Omega \left((v_1 - x_2)^2 + v_2^2\right) d\Omega + \frac{\alpha}{2}\int_\Omega \left(u_1^2 + u_2^2\right) d\Omega + \\ &+ 0.1 \int_\Omega \nabla \mathbf{v} : \nabla \mathbf{z} \, d\Omega - \int_\Omega p \nabla \cdot \mathbf{z} \, d\Omega + \\ &+ \int_\Omega r \nabla \cdot \mathbf{v} \, d\Omega - \int_\Omega \boldsymbol{f}(\mathbf{x},\mu) \cdot \mathbf{z} \, d\Omega - \int_\Omega \mathbf{u} \cdot \mathbf{z} \, d\Omega, \end{aligned} \tag{5.14}$$

then, the optimality condition is given by using the Fréchet differentiation:

$$\begin{aligned} \langle D_{\mathbf{v}} \mathcal{L}(\mathbf{v},\mathbf{z},\mathbf{u},p,r), \boldsymbol{\omega} \rangle &= \int_\Omega [(v_1 - x_2), v_2] \cdot \boldsymbol{\omega} \, d\Omega \\ &\quad - 0.1 \int_\Omega \boldsymbol{\omega} \cdot \Delta z d\Omega + \int_\Omega \boldsymbol{\omega} \cdot \nabla r = 0, \quad \forall \boldsymbol{\omega} \in Y^2, \\ \langle D_p \mathcal{L}(\mathbf{v},\mathbf{z},\mathbf{u},p,r), \beta \rangle &= \int_\Omega \beta \nabla \cdot \mathbf{z} \, d\Omega = 0 \quad \forall \beta \in Q, \\ \langle D_{\mathbf{u}} \mathcal{L}(\mathbf{v},\mathbf{z},\mathbf{u},p,r), \boldsymbol{\psi} \rangle &= \alpha \int_\Omega \mathbf{u} \cdot \boldsymbol{\psi} \, d\Omega - \int_\Omega \mathbf{z} \cdot \boldsymbol{\psi} \, d\Omega = 0 \quad \forall \boldsymbol{\psi} \in U^2, \\ \langle D_{\mathbf{z}} \mathcal{L}(\mathbf{v},\mathbf{z},\mathbf{u},p,r), \boldsymbol{\lambda} \rangle &= -0.1 \int_\Omega \boldsymbol{\lambda} \cdot \Delta \mathbf{v} \, d\Omega + \int_\Omega \boldsymbol{\lambda} \cdot \nabla p \, d\Omega \\ &\quad - \int_\Omega \boldsymbol{f} \cdot \boldsymbol{\lambda} \, d\Omega - \int_\Omega \mathbf{u} \cdot \boldsymbol{\lambda} \, d\Omega = 0 \quad \forall \boldsymbol{\lambda} \in Y^2, \\ \langle D_r \mathcal{L}(\mathbf{v},\mathbf{z},\mathbf{u},p,r), \rho \rangle &= \int_\Omega \rho \nabla \cdot \mathbf{v} \, d\Omega = 0 \quad \forall \rho \in Q. \end{aligned} \tag{5.15}$$

For the arbitrariness of the functions $\boldsymbol{\omega},\ \beta,\ \boldsymbol{\psi},\ \boldsymbol{\lambda},\ \rho$ we obtain the following system of equations

$$
\begin{cases}
-0.1\Delta \mathbf{z}(\mathbf{x},\mu) + \nabla r(\mathbf{x},\mu) = [x_2 - v_1(\mathbf{x},\mu),\ \ 0]^T \quad \text{in} \quad \Omega, \\
\nabla \cdot \mathbf{z}(\mathbf{x},\mu) = 0 \quad \text{in} \quad \Omega, \\
\alpha \mathbf{u}(\mathbf{x},\mu) = \mathbf{z}(\mathbf{x},\mu) \quad \text{in} \quad \Omega \\
-0.1\Delta \mathbf{v}(\mathbf{x},\mu) + \nabla p(\mathbf{x},\mu) = \boldsymbol{f}(\mathbf{x},\mu) + \mathbf{u}(\mathbf{x},\mu) \quad \text{in} \quad \Omega \\
\nabla \cdot \mathbf{v}(\mathbf{x},\mu) = 0 \quad \text{in} \quad \Omega \\
z_1(\mathbf{x},\mu) = z_2(\mathbf{x},\mu) = 0 \quad \text{on} \quad \Gamma_D \\
v_1(\mathbf{x},\mu) = x_2 \quad \text{and} \quad v_2(\mathbf{x},\mu) = 0 \quad \text{on} \quad \Gamma_D \\
-r(\mathbf{x},\mu) + 0.1\dfrac{\partial \mathbf{z}(\mathbf{x},\mu)}{\partial n} = 0 \quad \text{and} \ \ z_2(\mathbf{x},\mu) = 0 \quad \text{on} \ \ \Gamma_N, \\
-p(\mathbf{x},\mu) + 0.1\dfrac{\partial \mathbf{v}(\mathbf{x},\mu)}{\partial n} = 0 \quad \text{and} \quad v_2(\mathbf{x},\mu) = 0 \quad \text{on} \quad \Gamma_N.
\end{cases}
\tag{5.16}
$$

where the third equation is integrated in the PIARCH "step".

Regarding the parametric OCP of the Stokes Problem, we have six total settings included the baseline simulation run with the following hyper-parameters:

- Learning rate: 0.0005,
- Neurons in each hidden layer: [100,100,100,100],
- Optimizer: ADAM,
- Activation function: Softplus,
- Epochs of training: 50000,
- Sampling: $N_p = 400$ points latin hypercube-sampled in the physical domain within $\Omega$, $N_b = 1800$ points latin hypercube-sampled in the physical domain within $\partial\Omega$ and $N_{\boldsymbol{\mu}} = 10$ points latin hypercube-sampled in the parametric domain both for $\Omega$ and $\partial\Omega$.

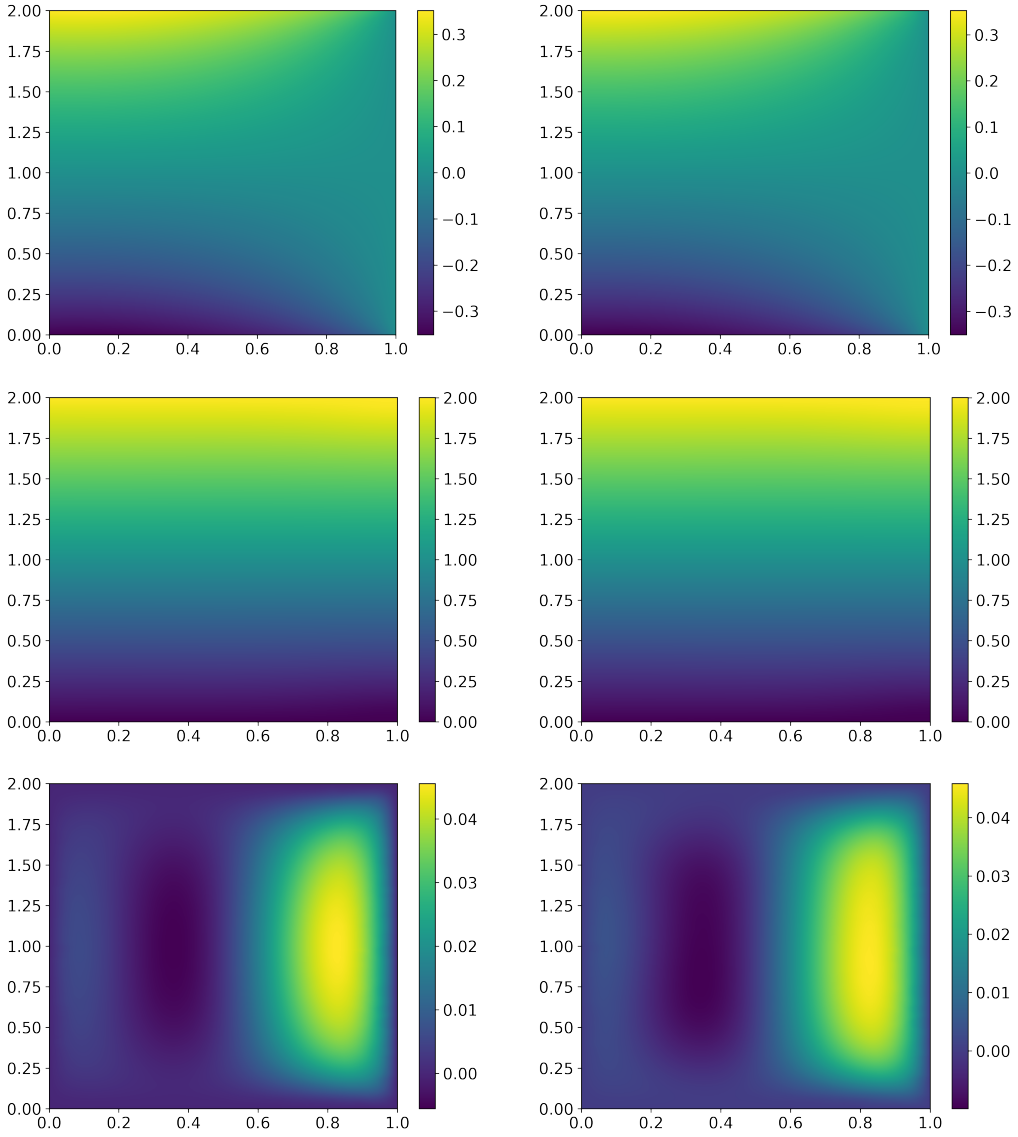



**Figure 5.5:** On the left we show the finite element solutions and on the right we show solutions of the PIARCH framework for: $p$, $v_1$, $v_2$, for $\mu = 0.05$.

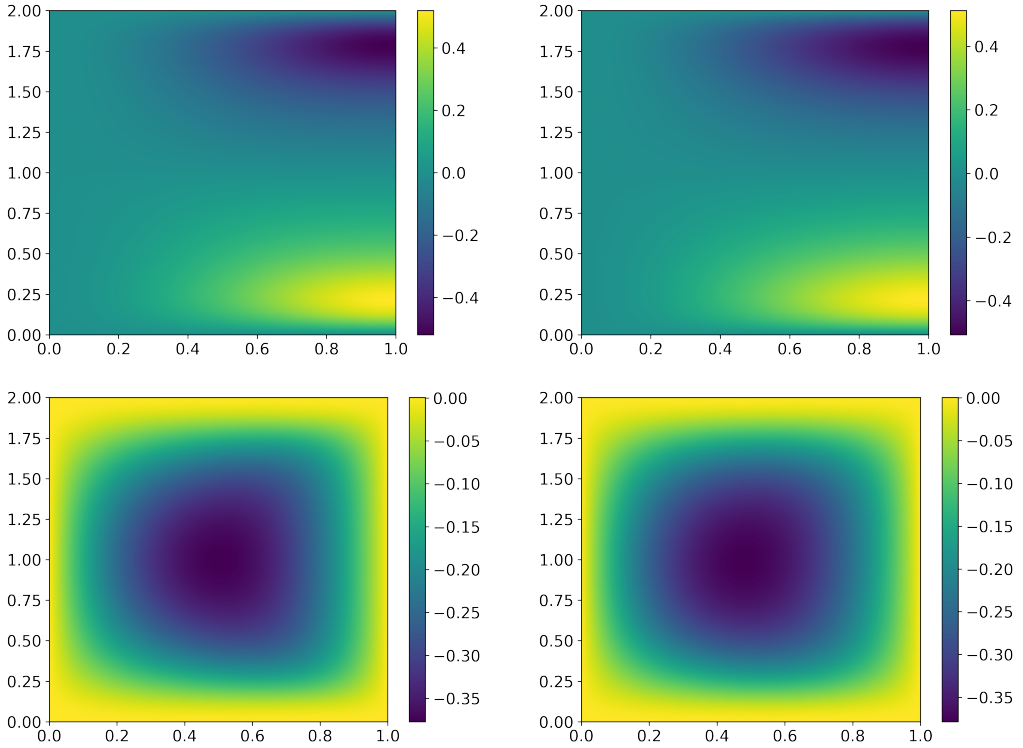



**Figure 5.6:** On the left we show the finite element solutions and on the right we show the solutions of the PIARCH framework for: $u_1$, $u_1$, for $\mu = 0.05$.

Concerning Tables 5.2, 5.3 the meaning of the settings used for training, such as "Grid sampling", is the same explained in 5.1. As we can see, the size of the network is increased to let it gain more expressivity, since the problem is more complex due to the number of variables to be predicted: before we had three, this time we have eight variables. Another important difference concerns the parametric space which is one-dimensional. This problem turned out to be more challenging than the previous for many reasons. First, we have many more conditions to satisfy, which means more possibilities of converging more to some of them and neglecting the others. The boundary conditions are no longer only Dirichlet, but, on the left, we have also Neumann boundary conditions, i.e. even if we enforce strong Dirichlet, we would not set to zero the error on that boundary. In fact, the relative errors are, generally, higher. This time the problems are more related to some particular variables that seem to be difficult to approximate such as $r$ and $v_2$.

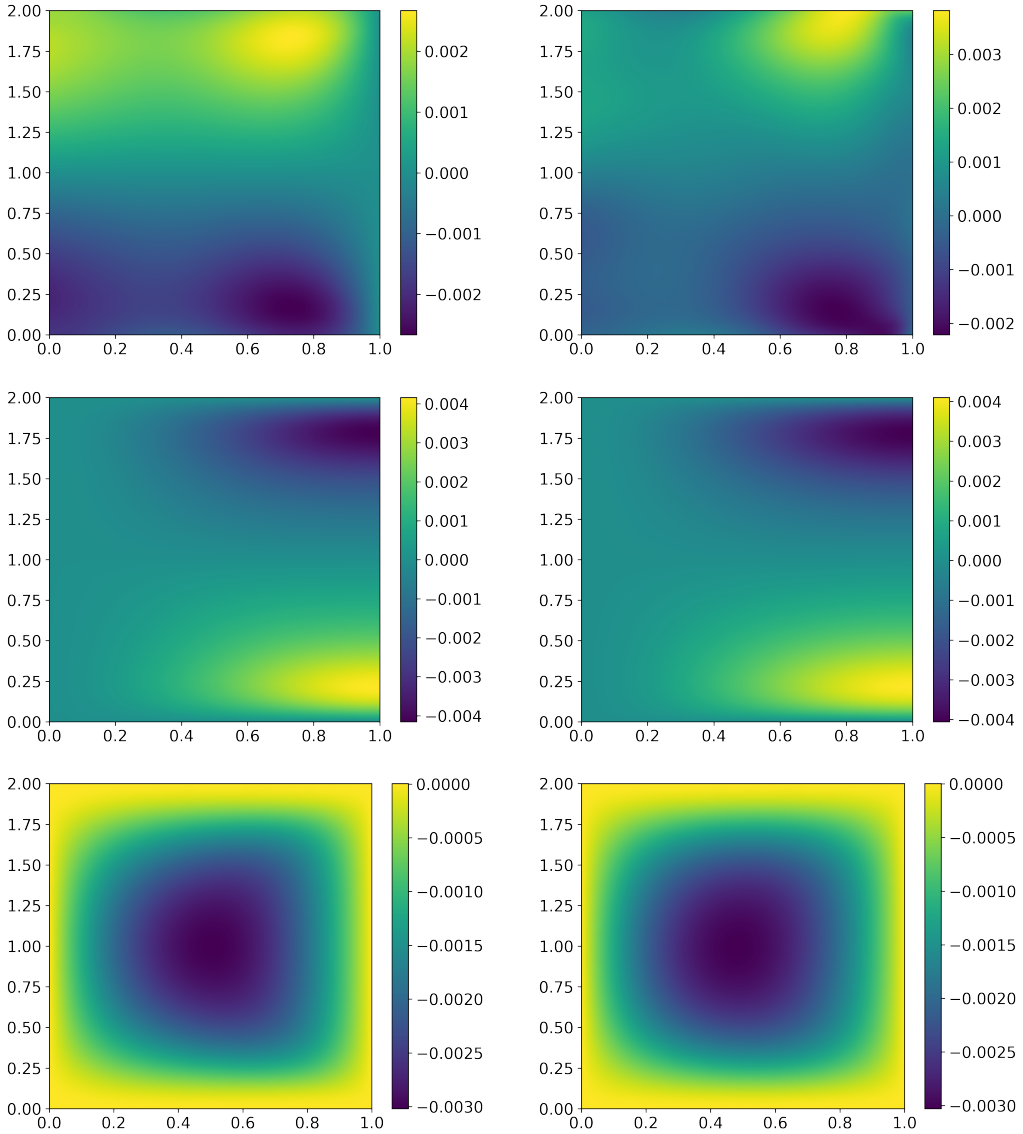



**Figure 5.7:** On the left we show the finite element solutions and on the right we show the solutions of the PIARCH framework for the adjoint variables in the following order: $r$, $z_1$, $z_2$, for $\mu = 0.05$.

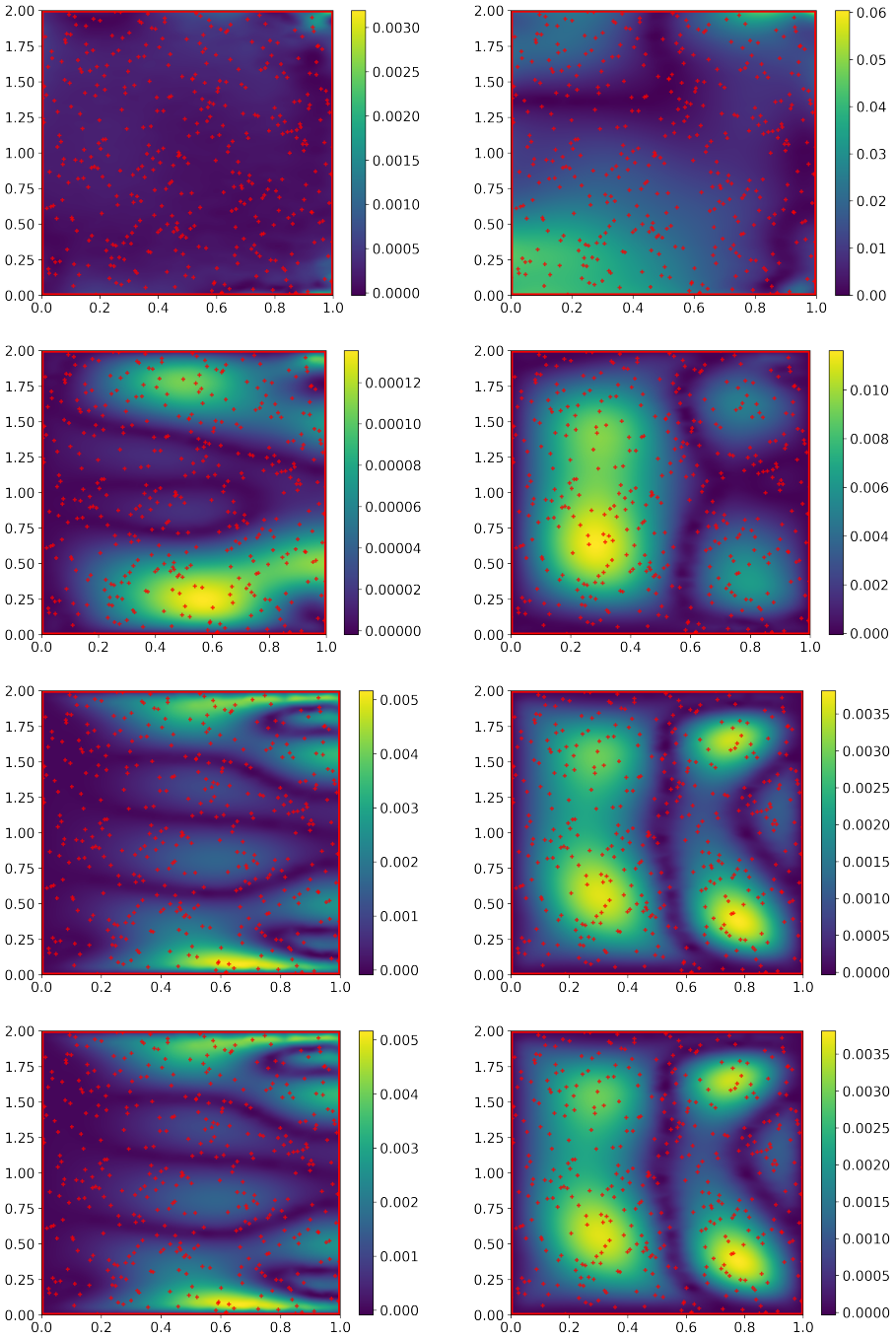


**Figure 5.8:** In the western reading direction we have the relative error for $p, r, v_1, v_2, u_1, u_2, z_1, z_2$ between the finite element solution and the PIARCH framework for $\mu = 0.05$. The red dots represent the training points used in the training phase.

Pressure state variable $p$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.03 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| Grid sampling | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| Strong BC | **0.01** | 0.01 | 0.01 | **0.009** | 0.01 | 0.02 |
| Strong + Grid | **0.01** | 0.007 | 0.007 | 0.01 | 0.009 | 0.01 |
| Chebyshev | 0.04 | 0.02 | 0.03 | 0.03 | 0.02 | 0.02 |
| Strong + Chebyshev | 0.02 | **0.006** | **0.005** | 0.02 | **0.007** | **0.005** |

Velocity state variable $v_1$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.005 | 0.009 | 0.01 | 0.007 | 0.009 | 0.01 |
| Grid sampling | 0.005 | 0.008 | 0.01 | 0.005 | 0.007 | 0.01 |
| Strong BC | **0.002** | **0.001** | **0.003** | **0.001** | **0.001** | **0.003** |
| Strong + Grid | **0.002** | **0.001** | **0.003** | 0.003 | 0.002 | 0.004 |
| Chebyshev | 0.01 | 0.01 | 0.02 | 0.008 | 0.008 | 0.01 |
| Strong + Chebyshev | 0.004 | 0.002 | 0.004 | 0.006 | 0.003 | 0.006 |

Velocity state variable $v_2$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.41 | 0.31 | 0.29 | 0.49 | 0.29 | 0.29 |
| Grid sampling | 0.31 | 0.2 | 0.27 | 0.41 | 0.23 | 0.23 |
| Strong BC | **0.19** | **0.07** | **0.06** | **0.13** | **0.06** | **0.07** |
| Strong + Grid | 0.24 | 0.11 | 0.08 | 0.22 | 0.11 | 0.1 |
| Chebyshev | 0.78 | 0.41 | 0.42 | 0.84 | 0.29 | 0.22 |
| Strong + Chebyshev | 0.34 | **0.07** | 0.08 | 0.54 | 0.16 | 0.16 |

Control variable $u_1$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.15 | 0.15 | 0.15 | 0.11 | 0.14 | 0.14 |
| Grid sampling | 0.1 | 0.13 | 0.15 | 0.11 | 0.12 | 0.13 |
| Strong BC | **0.06** | 0.04 | 0.06 | **0.04** | 0.05 | 0.07 |
| Strong + Grid | 0.07 | **0.03** | **0.05** | 0.07 | **0.03** | **0.06** |
| Chebyshev | 0.23 | 0.23 | 0.23 | 0.15 | 0.15 | 0.15 |
| Strong + Chebyshev | 0.08 | 0.03 | 0.05 | 0.11 | 0.04 | **0.06** |

**Table 5.2:** Relative error of the parametric stokes optimal control problem for different settings. The bold text refers to the lowest error between the ones in the column.

Control variable $u_2$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.16 | 0.11 | 0.09 | 0.15 | 0.10 | 0.09 |
| Grid sampling | 0.09 | 0.07 | 0.07 | 0.12 | 0.09 | 0.08 |
| Strong BC | **0.05** | **0.03** | **0.04** | **0.04** | **0.04** | **0.05** |
| Strong + Grid | 0.08 | 0.05 | **0.04** | 0.06 | 0.05 | **0.05** |
| Chebyshev | 0.22 | 0.16 | 0.15 | 0.19 | 0.11 | 0.1 |
| Strong + Chebyshev | 0.06 | **0.03** | **0.04** | 0.12 | 0.05 | **0.05** |

Adjoint pressure variable $r$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.94 | 0.6 | 0.75 | 0.67 | 0.50 | 0.66 |
| Grid sampling | **0.58** | 0.34 | 0.52 | 0.91 | 0.34 | 0.43 |
| Strong BC | 0.78 | 0.26 | 0.2 | **0.5** | **0.12** | **0.13** |
| Strong + Grid | 0.8 | 0.3 | 0.25 | 0.79 | 0.22 | 0.22 |
| Chebyshev | 0.89 | 0.70 | 1.03 | 1.04 | 0.48 | 0.48 |
| Strong + Chebyshev | 1.27 | **0.27** | **0.18** | 1.56 | 0.52 | 0.44 |

Adjoint velocity variable $z_1$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.64 | 0.31 | 0.36 | 0.11 | 0.14 | 0.14 |
| Grid sampling | 0.67 | 0.22 | 0.32 | 0.11 | 0.12 | 0.13 |
| Strong BC | **0.24** | 0.16 | 0.18 | **0.04** | 0.05 | 0.07 |
| Strong + Grid | 0.29 | 0.13 | 0.14 | 0.07 | **0.03** | **0.06** |
| Chebyshev | 0.78 | 0.44 | 0.46 | 0.15 | 0.15 | 0.15 |
| Strong + Chebyshev | 0.29 | **0.006** | **0.01** | 0.11 | 0.04 | **0.06** |

Adjoint velocity variable $z_2$

| | **PINN** | | | **PIARCH** | | |
|---|---|---|---|---|---|---|
| | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
| Baseline | 0.72 | 0.29 | 0.36 | 0.15 | 0.10 | 0.09 |
| Grid sampling | 0.69 | 0.27 | 0.35 | 0.12 | 0.09 | 0.08 |
| Strong BC | 0.15 | 0.06 | 0.06 | **0.04** | **0.04** | **0.05** |
| Strong + Grid | 0.11 | 0.05 | **0.04** | 0.06 | 0.05 | **0.05** |
| Chebyshev | 0.88 | 0.28 | 0.42 | 0.19 | 0.11 | 0.1 |
| Strong + Chebyshev | **0.1** | **0.04** | 0.05 | 0.12 | 0.05 | **0.05** |

**Table 5.3:** Relative error of the parametric stokes optimal control problem for different settings. The bold text refers to the lowest error between the ones in the column.

Overall the best framework is "Strong BC", remarking that enforcing strong BC is essential to obtain good approximation. However, also "Strong + Chebyshev" is good, outperforming "Strong BC" it many times. We can easily explain it: the complexity of the BC let us suppose that having more information nearby the boundary could be helpful in predicting them correctly. For example in Tables 5.2, 5.3 we can see that for many variables such as $u_y$ and $v_y$ there is high variability in the FEM solution, which is harder to approximate.

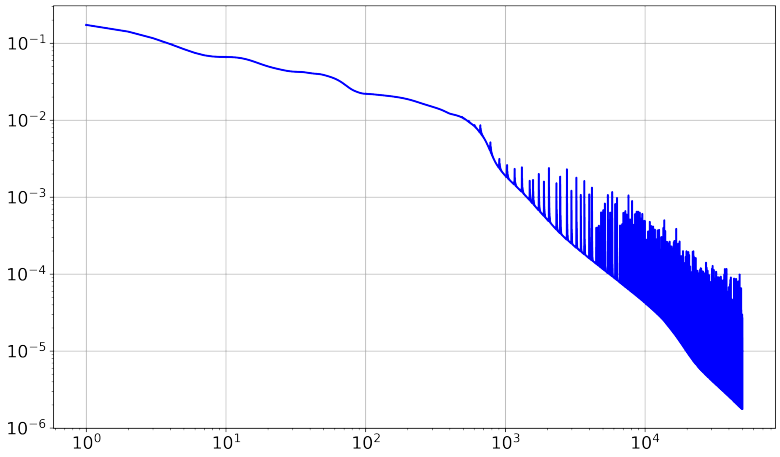



**Figure 5.9:** Trend of the mean of the loss functions of every equation for the Parametric Stokes OCP.

For the Stokes problem, as it can be seen in Figures 5.8, it is more evident the issue of predicting different output variables with different order of magnitude. In fact, the idea behind this work is that "classical" problems of the PINN may invalidate even more parametric OCP for the tendency of having more output, i.e. the control and also the adjoint variables.

# Chapter 6

# Fourier Embedding and Adaptive weights for Optimal Control Problems

In this section, we show the results obtained by enhancing the training with the two techniques we have seen previously: adaptive weights and FFE. In particular, we took the best setting for both parametric elliptic and parametric Stokes OCPs and trained with the enhancement of the two approaches discussed.

## 6.1 Adaptive weights

We now focus on the use of the Algorithm 1 and some issues related to it. From the numerical investigations, we found that the computations of the NTK, which is needed for the correct implementation of the algorithm, took too many resources in terms of computational time, making Algorithm 1 infeasible. Thus, for the sake of feasibility, we only computed the traces, and consequently the weights, at the beginning of the training and then used those values for all the epochs. This approach, even if related to a quite strong approximation assumption, can be reasonable when the NN trained is, as we have already said, over-parametrized.

Another important aspect is the aggregations of the residuals for the two optimization problems we considered.
Concerning the Parametric Elliptic OCP described by the system (5.8) and the boundary conditions (5.9), the residuals used for computing the NTK are the ones coming from the first and the last equation of (5.8).
All the other terms of the problem can be neglected thanks to strong boundary

conditions and the exploitation of the PIARCH architecture. Thus the weights of Algorithm 1 are related only to the loss of those two equations.

Concerning the Parametric Stokes OCP, the residuals taken into account for NTK are aggregated in the following way, referring to (5.16): the first two equations, which we can call it the physical equations, the fourth and the fifth equations, which we can call it the adjoint equations and the Neumann conditions. Thus we divided the residuals into three contributors, although the loss function minimized, apart from the weights, is the same. Thus, for this problem, the weights of Algorithm 1 were three. Namely:

$$\begin{aligned}
R_1(\mathbf{x},\mu) =& -0.1\Delta z_1(\mathbf{x},\mu) + \frac{\partial r(\mathbf{x},\mu)}{\partial x_1} - x_2 + v_1(\mathbf{x},\mu) \\
&+ -0.1\Delta z_2(\mathbf{x},\mu) + \frac{\partial r(\mathbf{x},\mu)}{\partial x_2} + \nabla\cdot\mathbf{z}(\mathbf{x},\mu), \\
R_2(\mathbf{x},\mu) =& -0.1\Delta v_1(\mathbf{x},\mu) + \frac{\partial p(\mathbf{x},\mu)}{\partial x_1} - \boldsymbol{f}_1(\mathbf{x},\mu) - u_1(\mathbf{x},\mu) \\
&- 0.1\Delta v_2(\mathbf{x},\mu) + \frac{\partial p(\mathbf{x},\mu)}{\partial x_2} - \boldsymbol{f}_2(\mathbf{x},\mu) - u_2(\mathbf{x},\mu) + \nabla\cdot\mathbf{v}(\mathbf{x},\mu), \\
R_3(\mathbf{x},\mu) =& -r(\mathbf{x},\mu) + 0.1\frac{\partial \mathbf{z}(\mathbf{x},\mu)}{\partial n} - p(\mathbf{x},\mu) + 0.1\frac{\partial \mathbf{v}(\mathbf{x},\mu)}{\partial n}.
\end{aligned} \tag{6.1}$$

As another possible choice, one could take each equation separately and compute a specific weight for each one.

Taking into account the last considerations, we computed also the eigenvalues of the NTK of the two problems in Figure 6.1. To speed up computations, they were computed using a NN with one layer with 512 neurons and a reduced number of sample points. It is easy to observe that the majority of the eigenvalues is close to zero, with a small amount of them being high in magnitude. In particular, the number of eigenvalues greater than 1 is 170 out of 3042 for the elliptic constrained problem and 19 out of 3000 for the Stokes constrained. This disparity, as already mentioned, is the basis of SB: in fact, ideally, we would like to have all the eigenvalues take high values.

In Table 6.1, we show the relative errors for parametric elliptic OCP using the best setting found during the previous experiment, which, in this case, was the "Strong Grid" setting. We used the same hyper-parameters and enhanced the training as we said earlier.
The same is true for Table 6.2 concerning parametric Stokes OCP, where the best setting was "Strong".

| variable | $\mu = 0.01$ | $\mu = 0.1$ | $\mu = 1$ |
|---|---|---|---|
| $u$ | 1.0 | 1.0 | 0.99 |
| $y$ | 0.19 | 1.27 | 15.75 |
| $z$ | 1.0 | 1.0 | 0.99 |

**Table 6.1:** Relative errors of Parametric elliptic OCP using the adaptive weights in Algorithm 1.

| variable | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
|---|---|---|---|
| $v_x$ | 0.37 | 0.36 | 0.34 |
| $v_y$ | 1.95 | 0.96 | 0.80 |
| $u_x$ | 0.98 | 0.98 | 0.98 |
| $u_y$ | 0.74 | 0.77 | 0.79 |
| $p$ | 0.95 | 0.93 | 0.92 |
| $r$ | 26.63 | 18.7 | 19.8 |
| $z_x$ | 0.98 | 0.98 | 0.98 |
| $z_y$ | 0.73 | 0.77 | 0.79 |

**Table 6.2:** Relative errors of Parametric Stokes OCP using the adaptive weights in Algorithm 1.

All the results obtained are disappointing in terms of accuracy and do not enhance the training of the PINN. This is because the numerator of the weights $\beta_j$ of Algorithm 1 are equal to the trace of the NTK, which, in general, can be a very big number as it can be seen in Figure 6.1, where we computed the eigenvalues of the NTK for both problems and put them in the ascending order. And since the trace is constant, in this case, it is equivalent to having a different, much bigger, learning rate.

| variable | $\mu = 0.01$ | $\mu = 0.1$ | $\mu = 1$ |
|---|---|---|---|
| $u$ | 0.10 | 0.03 | 0.003 |
| $y$ | 0.04 | 0.06 | 0.02 |
| $z$ | 0.10 | 0.03 | 0.003 |

**Table 6.3:** Relative errors of Parametric elliptic OCP using the adaptive weights in Algorithm 1 and a decreased learning rate.

| variable | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
|---|---|---|---|
| $v_x$ | 0.007 | 0.005 | 0.009 |
| $v_y$ | 0.61 | 0.28 | 0.21 |
| $u_x$ | 0.17 | 0.07 | 0.13 |
| $u_y$ | 0.18 | 0.09 | 0.1 |
| $p$ | 0.09 | 0.02 | 0.02 |
| $r$ | 1.78 | 0.37 | 0.53 |
| $z_x$ | 0.17 | 0.07 | 0.13 |
| $z_y$ | 0.18 | 0.09 | 0.1 |

**Table 6.4:** Relative errors of Parametric Stokes OCP using the adaptive weights in Algorithm 1 and a decreased learning rate.

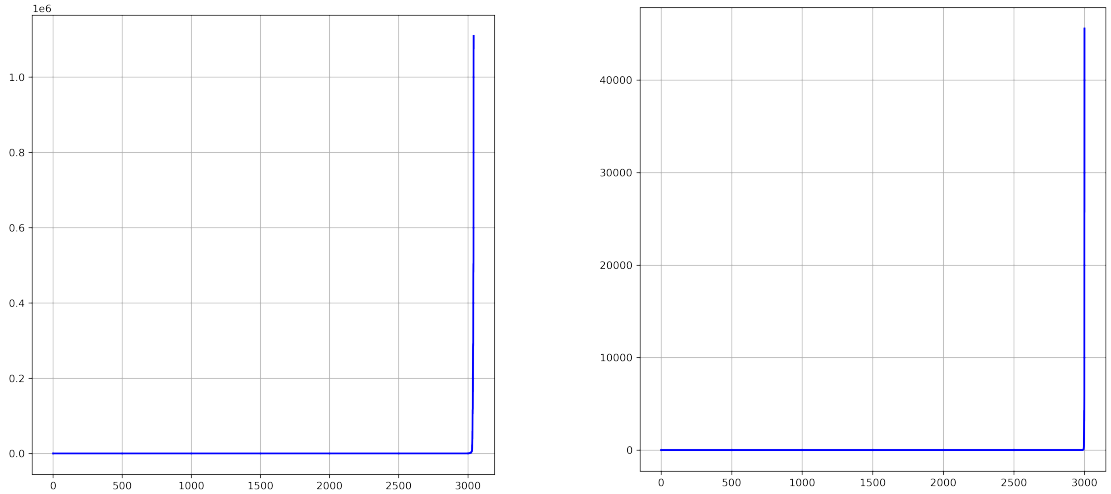



**Figure 6.1:** Eigenvalues of the NTK plotted in ascending order. On the left, eigenvalues of Parametric Elliptic OCP are showed. On the right, eigenvalues of Parametric Stokes OCP are showed.

For this reason, we trained again the PINN with a learning rate equal to $\alpha = 0.00005$, i.e. one order of magnitude less. Results are summarized in Tables 6.3 and 6.4. In this case, the behavior found seems reasonable and possibly promising. Comparing the performance with respect to the best settings already found, we can say that, although we did not enhance the performances obtained in the previous chapters, we showed that Adaptive weights can be used for parametric OCPs. If a training fully respecting Algorithm 1 was performed, we could have given a more precise and robust opinion on that. In general, to explore some methods that could make the computation of the traces of the NTK easier is an interesting direction of research.

Thus a better study of this approach could be conducted, especially when dealing with parametric OCPs, where the equation to satisfy are multiple and coupled.

## 6.2 Fourier Feature Embeddings

Concerning FFE, again we trained the best setting found for the two problems and tried to enhance it with FFE. We conducted three experiments, all with an expansion of the feature of a factor $m = 100$, with the following combinations of values:

- $\tau_1 = 1$ and $\tau_2 = 5$.
- $\tau_1 = 1$ and $\tau_2 = 10$.
- $\tau_1 = 5$ and $\tau_2 = 10$.

| variable | $\mu = 0.01$ | $\mu = 0.1$ | $\mu = 1$ |
|---|---|---|---|
| $u$ | 0.96 | 0.9 | 14.89 |
| $y$ | 0.96 | 0.92 | 12.03 |
| $z$ | 0.96 | 0.9 | 14.89 |

**Table 6.5:** Relative errors of Parametric elliptic OCP using FFE with $\tau_1 = 1$ and $\tau_2 = 5$.

| variable | $\mu = 0.01$ | $\mu = 0.1$ | $\mu = 1$ |
|---|---|---|---|
| $u$ | 0.95 | 0.84 | 0.18 |
| $y$ | 0.95 | 0.85 | 0.13 |
| $z$ | 0.95 | 0.84 | 0.18 |

**Table 6.6:** Relative errors of Parametric elliptic OCP using FFE with $\tau_1 = 5$ and $\tau_2 = 10$.

The Tables 6.5, 6.6, and 6.7 summarize the results obtained applying FFE to the Parametric Elliptic OCP. As we can see no improvements were made with respect to the best setting found earlier. This underlines the main disadvantage of FFE, which is the difficulty of tuning the parameter $\tau$.
However, it is interesting to see that the setting with $\tau_1 = 5$ and $\tau_1 = 10$ performed better than the others. This suggests that, in this problem, higher values of $\tau$ are more useful for detecting variations in the solution. Keeping in mind this, one could probably improve these results by performing a greater expansion of the

| variable | $\mu = 0.01$ | $\mu = 0.1$ | $\mu = 1$ |
|---|---|---|---|
| $u$ | 0.93 | 0.81 | 2.06 |
| $y$ | 0.94 | 0.85 | 1.59 |
| $z$ | 0.93 | 0.81 | 2.06 |

**Table 6.7:** Relative errors of Parametric elliptic OCP using FFE with $\tau_1 = 1$ and $\tau_2 = 10$.

| variable | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
|---|---|---|---|
| $v_x$ | 0.005 | 0.004 | 0.01 |
| $v_y$ | 0.77 | 0.33 | 0.27 |
| $u_x$ | 0.15 | 0.09 | 0.17 |
| $u_y$ | 0.28 | 0.12 | 0.1 |
| $p$ | 0.025 | 0.013 | 0.025 |
| $r$ | 0.77 | 0.32 | 0.62 |
| $z_x$ | 0.15 | 0.09 | 0.17 |
| $z_y$ | 0.28 | 0.12 | 0.1 |

**Table 6.8:** Relative errors of Parametric Stokes OCP using the FFE with $\tau_1 = 1$ and $\tau_2 = 5$.

input (in this context we expanded to 50 for each embedding).

In Tables 6.8, 6.9, and 6.10 are summarized results for the Parametric Stokes OCP. Similar considerations can be made, noticing that, in this problem, lower values of $\tau$ seem to deliver a better performance.

| variable | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
|---|---|---|---|
| $v_x$ | 0.007 | 0.01 | 0.02 |
| $v_y$ | 1.25 | 1.0 | 0.97 |
| $u_x$ | 1.0 | 1.0 | 1.0 |
| $u_y$ | 1.0 | 1.0 | 1.0 |
| $p$ | 1.03 | 1.02 | 1.01 |
| $r$ | 1.0 | 1.0 | 1.0 |
| $z_x$ | 1.0 | 1.0 | 1.0 |
| $z_y$ | 1.0 | 1.0 | 1.0 |

**Table 6.9:** Relative errors of Parametric Stokes OCP using the FFE with $\tau_1 = 5$ and $\tau_2 = 10$.

| variable | $\mu = 0.5$ | $\mu = 1$ | $\mu = 1.5$ |
|---|---|---|---|
| $v_x$ | 0.006 | 0.006 | 0.02 |
| $v_y$ | 0.9 | 0.36 | 0.37 |
| $u_x$ | 0.24 | 0.14 | 0.22 |
| $u_y$ | 0.31 | 0.14 | 0.12 |
| $p$ | 0.034 | 0.02 | 0.03 |
| $r$ | 1.07 | 0.54 | 0.85 |
| $z_x$ | 0.24 | 0.14 | 0.22 |
| $z_y$ | 0.31 | 0.14 | 0.12 |

**Table 6.10:** Relative errors of Parametric Stokes OCP using the FFE with $\tau_1 = 1$ and $\tau_2 = 10$.

# Conclusions

The purpose of this thesis was to improve the performance of the PINN for parametric OCPs. First, we studied some aspects of the net implementation such as the boundary conditions imposition and the sampling techniques, but we also validated the PIARCH architecture already proposed in [26]. From this, we obtained satisfying results, comparable to the ground truth solution based on a Finite Element solver. Afterward, we exploited two techniques: FFE, and Algorithm 1 to rediced the SB in PINN training. The two strategies have been successfully tested on a toy problem. Finally, we applied FFE and Algorithm 1 to Parametric OCPs, which, to the best of our knowledge, was never done before. This preliminary numerical investigation shows some criticalities.
Concerning FFE, one could obtain better results with a fine-tuning of the hyperparameter $\tau$ which, as we showed for some simple one-dimensional problems, can influence the training a lot. Another interesting direction for improvement lies in using an embedding with a greater number of features. This could be the key ingredient to increase the accuracy of the training, especially for the Stokes problem which is more complex indeed.

Concerning the adaptive weights, performances were similar, although slightly worse, with respect to the ones found in 5.1 and 5.2 when applying the Algorithm 1. Despite this behavior, we think that this approach should be deeply investigated in order to show some improvements with respect to standard PINN training. We only computed the weights at the start of the training and kept them constant, which, at times, could be a strong assumption and might yield inaccurate training. We recall that the computation of the NTK is a time-consuming activity and the approach is infeasible with the provided computational resources, unless some ad hoc strategies are developed. Potentially, the Adaptive weights could be a very good method. First, the number of conditions in OCPs are so many that the SB may likely occur, and, second, it is an approach that is not problem-dependent, because it relies on NTK theory, more specifically on the eigenvalues which describe the convergence of the PINN to the solution of the problem.

# Bibliography


[1] Juan De los Reyes and Fredi Tröltzsch. «Optimal Control of the Stationary Navier-Stokes Equations with Mixed Control-State Constraints». In: *SIAM J. Control Optim.* 46 (Jan. 2007), pp. 604–629. DOI: 10.1137/050646949 (cit. on p. 1).

[2] Luca Dede. «Optimal flow control for Navier-Stokes equations: Drag minimization». In: *International Journal for Numerical Methods in Fluids* 55 (Oct. 2007), pp. 347–366. DOI: 10.1002/fld.1464 (cit. on p. 1).

[3] Federico Negri, Andrea Manzoni, and Gianluigi Rozza. «Reduced basis approximation of parametrized optimal flow control problems for the Stokes equations». In: *Computers and Mathematics with Applications* 69.4 (2015), pp. 319–336. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2014.12.010. URL: https://www.sciencedirect.com/science/article/pii/S0898122114006075 (cit. on p. 1).

[4] Maria Strazzullo, Francesco Ballarin, Renzo Mosetti, and Gianluigi Rozza. «Model Reduction for Parametrized Optimal Control Problems in Environmental Marine Sciences and Engineering». In: *SIAM Journal on Scientific Computing* 40.4 (Jan. 2018), B1055–B1079. ISSN: 1095-7197. DOI: 10.1137/17m1150591. URL: http://dx.doi.org/10.1137/17M1150591 (cit. on p. 1).

[5] Gianluigi Rozza, Andrea Manzoni, and F. Negri. «Reduction strategies for PDE-constrained optimization problems in haemodynamics». In: *ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers* (Jan. 2012), pp. 1749–1768 (cit. on p. 1).

[6] René Pinnau, Claudia Totzeck, and Oliver Tse. «The Quasi-Neutral Limit in Optimal Semiconductor Design». In: *SIAM Journal on Control and Optimization* 55.4 (Jan. 2017), pp. 2603–2635. ISSN: 1095-7138. DOI: 10.1137/15m1051877. URL: http://dx.doi.org/10.1137/15M1051877 (cit. on p. 1).

[7] Eduard Bader, Mark Kärcher, Martin A. Grepl, and Karen Veroy. «Certified Reduced Basis Methods for Parametrized Distributed Elliptic Optimal Control Problems with Control Constraints». In: *SIAM Journal on Scientific*

*Computing* 38.6 (2016), A3921–A3946. DOI: 10.1137/16M1059898 (cit. on p. 1).

[8] Luca Dede'. «Adaptive and Reduced Basis methods for optimal control problems in environmental applications». PhD thesis. Politecnico di Milano, 2008. URL: https://infoscience.epfl.ch/handle/20.500.14299/84327 (cit. on p. 1).

[9] Mark Kärcher and Martin A. Grepl. «A certified reduced basis method for parametrized elliptic optimal control problems». In: *ESAIM: Control, Optimisation and Calculus of Variations* 20.2 (2014), pp. 416–441. DOI: 10.1051/cocv/2013069 (cit. on p. 1).

[10] K. Kunisch and S. Volkwein. «Control of the Burgers equation by a reduced-order approach using proper orthogonal decomposition». In: *J. Optim. Theory Appl.* 102.2 (1999), pp. 345–371. ISSN: 0022-3239. DOI: 10.1023/A:1021732508059 (cit. on p. 1).

[11] Hinze Michael, Kutz J. Nathan, Mula Olga, and Urban Karsten. *Model Order Reduction and Applications*. Vol. 22. Computational Science and Engineering. Springer, 2021. ISBN: 978-3-030-72983-6. DOI: 10.1007/978-3-030-72983-6. URL: https://doi.org/10.1007/978-3-030-72983-6 (cit. on pp. 1, 8).

[12] M. Raissi, P. Perdikaris, and G.E. Karniadakis. «Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations». In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045. URL: https://www.sciencedirect.com/science/article/pii/S0021999118307125 (cit. on pp. 2, 11).

[13] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. «Physics-Informed Neural Networks for Heat Transfer Problems». In: *Journal of Heat Transfer* 143.6 (Apr. 2021), p. 060801. ISSN: 0022-1481. DOI: 10.1115/1.4050542. eprint: https://asmedigitalcollection.asme.org/heattransfer/article-pdf/143/6/060801/6688635/ht_143_06_060801.pdf. URL: https://doi.org/10.1115/1.4050542 (cit. on p. 2).

[14] Ameya Jagtap, Ehsan Kharazmi, and George Karniadakis. «Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems». In: *Computer Methods in Applied Mechanics and Engineering* 365 (June 2020), p. 113028. DOI: 10.1016/j.cma.2020.113028 (cit. on p. 2).

[15] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. «NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations». In: *Journal of Computational Physics* 426 (Feb. 2021), p. 109951. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2020.109951. URL: http://dx.doi.org/10.1016/j.jcp.2020.109951 (cit. on p. 2).

[16] Ehsan Kharazmi, Zhongqiang Zhang, and George E.M. Karniadakis. «hp-VPINNs: Variational physics-informed neural networks with domain decomposition». In: *Computer Methods in Applied Mechanics and Engineering* 374 (2021), p. 113547. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2020.113547. URL: https://www.sciencedirect.com/science/article/pii/S0045782520307325 (cit. on p. 2).

[17] Mohammadamin Mahmoudabadbozchelou, Marco Caggioni, Setareh Shahsavari, William Hartt, George Karniadakis, and Safa Jamali. «Data-driven physics-informed constitutive metamodeling of complex fluids: A multifidelity neural network (MFNN) framework». In: *Journal of Rheology* 65 (Mar. 2021), pp. 179–198. DOI: 10.1122/8.0000138 (cit. on p. 2).

[18] Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. «PPINN: Parareal physics-informed neural network for time-dependent PDEs». In: *Computer Methods in Applied Mechanics and Engineering* 370 (2020), p. 113250. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2020.113250. URL: https://www.sciencedirect.com/science/article/pii/S0045782520304357 (cit. on p. 2).

[19] G. Pang, M. D'Elia, M. Parks, and G.E. Karniadakis. «nPINNs: Nonlocal physics-informed neural networks for a parametrized nonlocal universal Laplacian operator. Algorithms and applications». In: *Journal of Computational Physics* 422 (2020), p. 109760. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2020.109760. URL: https://www.sciencedirect.com/science/article/pii/S0021999120305349 (cit. on p. 2).

[20] Liu Yang, Xuhui Meng, and George Em Karniadakis. «B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data». In: *Journal of Computational Physics* 425 (2021), p. 109913. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2020.109913. URL: https://www.sciencedirect.com/science/article/pii/S0021999120306872 (cit. on p. 2).

[21] Sifan Wang, Xinling Yu, and Paris Perdikaris. «When and Why PINNs Fail to Train: A Neural Tangent Kernel Perspective». In: *Journal of Computational Physics* 449 (2022), p. 110768. DOI: 10.1016/j.jcp.2021.110768. (Visited on 09/27/2022) (cit. on pp. 2, 16–18, 21).

[22] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. *An Expert's Guide to Training Physics-informed Neural Networks*. 2023. arXiv: 2308.08468 [cs.LG]. URL: https://arxiv.org/abs/2308.08468 (cit. on p. 2).

[23] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. *On the Spectral Bias of Neural Networks*. 2019. arXiv: 1806.08734 [stat.ML]. URL: https://arxiv.org/abs/1806.08734 (cit. on p. 2).

[24] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. arXiv: 2006.10739 [cs.CV]. URL: https://arxiv.org/abs/2006.10739 (cit. on pp. 2, 19).

[25] Sifan Wang, Hanwen Wang, and Paris Perdikaris. «On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks». In: *Computer Methods in Applied Mechanics and Engineering* 384 (2021), p. 113938. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.113938. URL: https://www.sciencedirect.com/science/article/pii/S0045782521002759 (cit. on pp. 2, 19, 20).

[26] Nicola Demo, Maria Strazzullo, and Gianluigi Rozza. «An Extended Physics Informed Neural Network for Preliminary Analysis of Parametric Optimal Control Problems». In: *Computers & Mathematics with Applications* 143 (2023), pp. 383–396. DOI: 10.1016/j.camwa.2023.05.004. (Visited on 05/06/2024) (cit. on pp. 2, 24, 33, 57).

[27] Alfio Quarteroni. *Numerical Models for Differential Problems*. 2nd. Springer Publishing Company, Incorporated, 2013. ISBN: 8847055210 (cit. on pp. 4, 8, 9).

[28] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE Constraints*. Mathematical Modelling: Theory and Applications. Springer Netherlands, 2008. ISBN: 9781402088391. URL: https://books.google.it/books?id=PFbqxa2uDS8C (cit. on p. 7).

[29] Sorin Micu. «Uniform Boundary Controllability of a Semidiscrete 1-D Wave Equation with Vanishing Viscosity». In: *SIAM J. Control and Optimization* 47 (Jan. 2008), pp. 2857–2885. DOI: 10.1137/070696933 (cit. on p. 8).

[30] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. «Physics-informed neural networks for inverse problems in nano-optics and metamaterials». In: *Optics Express* 28.8 (Apr. 2020), p. 11618. ISSN: 1094-4087. DOI: 10.1364/oe.384875. URL: http://dx.doi.org/10.1364/OE.384875 (cit. on p. 11).

[31] Ameya D. Jagtap, Zhiping Mao, Nikolaus Adams, and George Em Karniadakis. «Physics-informed neural networks for inverse problems in supersonic flows». In: *Journal of Computational Physics* 466 (Oct. 2022), p. 111402. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.111402. URL: http://dx.doi.org/10.1016/j.jcp.2022.111402 (cit. on p. 11).

[32] Claudio Canuto. *Model Order Reduction and Machine Learning, Lecture Notes*. Lecture notes from Politecnico di Torino, March-June 2023. 2023 (cit. on p. 13).

[33] Sebastian Barschkis. *Exact and soft boundary conditions in Physics-Informed Neural Networks for the Variable Coefficient Poisson equation*. 2023. arXiv: 2310.02548 [cs.LG]. URL: https://arxiv.org/abs/2310.02548 (cit. on p. 24).