POLITECNICO DI TORINO

Dipartimento di Scienze Matematiche

Corso di Laurea Magistrale in Ingegneria Matematica

Tesi di Laurea Magistrale

On the application of PINNs to Mathematical Biology problems: a comparative study

Relatori

Prof. L. PREZIOSI

Prof. N. SFAKIANAKIS

Candidata

Giulia RUBINI

Luglio 2026

Abstract

Physics-Informed Neural Networks (PINNs) have emerged as a promising meshless framework for solving partial differential equations, embedding physical laws directly into the training process of neural networks. Their potential applicability to mathematical biology is particularly appealing, as many biological phenomena are governed by complex, nonlinear PDEs defined on irregular domains where classical discretisation methods can be costly or cumbersome. This thesis investigates the capabilities and limitations of PINNs when applied to biological equations, with a focus on systems that exhibit spatial pattern formation. Following a thorough theoretical treatment of the PINN framework, including loss formulation, network architecture, training dynamics, and known failure modes, two case studies of increasing complexity are examined. First, the one-dimensional Burgers equation is solved using a standard PINN, yielding accurate results that validate the methodology as a reliable solver for well-posed, low-dimensional nonlinear problems. Second, the two-dimensional Gray-Scott reaction-diffusion system is tackled using both a vanilla PINN and a hard-constraint PINN formulation. Reference solutions are produced via the Finite Element Method and the Finite Volume Method to serve as ground truth. The vanilla PINN fails to satisfy initial conditions and produces qualitatively incorrect solutions. The hard-constraint manages to enforce exactly the initial conditions, but proves unable to capture the emergence and evolution of spatial patterns in subsequent time steps. These results expose fundamental limitations of current PINN methodologies when confronted with stiff, multi-scale reaction-diffusion dynamics, including spectral bias, loss imbalance, and sensitivity to the unstable equilibria that drive pattern formation. At the same time, they delineate a clear and applicable research plan for hard-constraint PINNs: advances in adaptive loss weighting, time-domain decomposition, and frequency-aware architectures hold genuine promise for overcoming these barriers. As these methods advance, PINNs could become a powerful complement to classical solvers in the study of biological processes.

Table of Contents

List of Figures

Chapter 1

Introduction

Solving Partial Differential Equations (PDEs) is a fundamental task in computational sciences. PDEs arise in numerous scientific areas, for example in the modelling of biological processes, such as population dynamics, tumour growth and pattern formation. Analytical solutions are not available for most such equations, and computational methods are therefore employed to find an approximate solution. The main traditional numerical methods rely on the discretization of the domain in a mesh, to be able to approximate derivatives. The final solution is usually calculated on a grid, that can be coarser or finer depending on the requirements of the problem. The advent of Machine Learning has sparked interest in a novel methodology to solve PDEs, using data-driven neural networks. In particular, the most successful method to date is the Physics-Informed Neural Network (PINN). PINNs encode information about physical processes into the learning algorithm, in order to steer the neural network towards the solution. With the use of automatic differentiation, it is possible to encode any physical law expressed as a PDE into the loss function of a neural network. The main innovation of PINNs, compared to traditional numerical solvers, is that they are mesh-free, and therefore easier to implement for more complex domain shapes. Despite this great advantage, PINNs still face challenges such as sensitivity to parameters' choice, and convergence to local minima of the objective function.

This work provides an overview of traditional methods and of PINNs state of the art, and two case studies that exhibit the performance of PINNs in biologically relevant problems. The thesis is structured as follows: the first chapter provides a background introduction to Partial Differential Equations and classical methods to solve them. The second chapter presents the state of the art of Physics-Informed Neural network, explaining the mechanism of this methodology. This chapter also lays out an overview of the main challenges that the use of PINNs as solvers for PDEs currently faces, providing references to relevant works. The last section of the chapter focuses on the difference between vanilla PINNs and hard-constrained PINNs. The former incorporate initial and boundary conditions as soft penalty terms in the loss function, seeking to minimize the discrepancy between the prescribed and predicted values at a set of collocation points, while the latter satisfy the conditions exactly by

embedding them directly into the network architecture. Focus on this distinction is relevant as the enforcement of exact initial conditions is essential for the simulation of time-evolving patterns. The third chapter exhibits the results of two case studies of increasing difficulty: the solution of the one-dimensional Burgers' equation and the solution of the two-dimensional Gray-Scott system of equations. Results highlight both the successful performance in the first case study, and the challenges that PINNs face when solving stiff problems with bigger spatio-temporal domains. The use of both Vanilla and Hard PINNs underlines the importance of initial conditions enforcement. Finally, the last chapter summarizes the results acquired and considers possible future areas of research for the advancement of PINNs as PDE solvers.

Chapter 2

Background

2.1 PDEs

Partial Differential Equations (PDEs) describe the time and space evolution of a certain quantity, for example concentration of a chemical species, density of a population or pressure of a fluid. PDEs arise from the modelling of a great variety of processes in different areas of study, such as physics, mathematics, economics, and biology.

The first studies on PDEs began in the 18th century, when this type of equation emerged in the modelling of continuous media in classical problems, such as the vibrating string model or the heat conduction model. Between the end of the 18th century and the beginning of the 19th century, three important PDEs were already defined: the *heat equation*, the Laplace equation and the *wave equation*, which later became the main examples of second-order PDEs, defining also the classification in elliptic, hyperbolic and parabolic equations.

The first models that used PDEs in biology appeared in the 1900s, and in the 1930s they were applied to describe the spreading of diseases and genes. In the 1950s there were further important developments; among these, Turing's work on pattern formation [1].

A partial differential equation of order k is an expression of the form:

$$F(D^k u(x), D^{k-1} u(x), ..., Du(x), u(x), x) = 0 \ , \qquad x \in U$$

where the operator F is a given function and $u : U \to \mathbb{R}$ is unknown. Finding a solution to a partial differential equation means finding u such that it verifies the equation and satisfies certain initial and boundary conditions.

PDEs can be classified in different ways. An important distinction has to be made considering linear or non-linear PDEs, since non linearity adds complexity to the problem. Non-linear equations are common in most real-life applications. One possible way to classify different PDEs is that of dividing them in elliptic, parabolic and hyperbolic equations.

Considering a generic second-order partial differential equation, it can be written as:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + \text{lower order terms} = 0 \tag{2.1}$$

Then, the distinction in the three classes is based on the sign of the determinant $B^2 - 4AC$, that is determined by the sign of the coefficient matrix's eigenvalues.

When the determinant is negative the equation is said to be an elliptic partial differential equation. The eigenvalues all take the same sign. These equations are time-independent and they represent steady state systems. Information propagates instantly in the whole domain, so they are charcterized by an infinite propagation speed. The main example of this class is the Laplace equation:

$$\Delta u = 0 \qquad \text{where} \qquad \Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{2.2}$$

When the determinant is equal to zero (or there is at least one zero eigenvalue) the equation is a parabolic equation, the main example of which is the *heat equation* :

$$\frac{\partial u}{\partial t} = \Delta u \tag{2.3}$$

The problems described by this class of equations are diffusion-driven problems. A notable example of parabolic systems is that of reaction-diffusion equations, which couple the diffusion operator with nonlinear reaction terms and are capable of producing complex spatial patterns, as in the case of the Gray-Scott model discussed later in this work.

The last case is that of hyperbolic equations. Equations pertaining to this class have a positive determinant, which implies that there are both positive and negative eigenvalues. The main example is the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = \Delta u \tag{2.4}$$

The phenomena that are described by this class of equations are characterized by a wave-like behaviour, where energy is conserved and discontinuities are propagated at a finite-speed along characteristic curves.

A partial differential equation, or a system of them, has to be coupled to initial and boundary conditions in order to be a well-posed problem, with a unique solution.

2.2 Classical methods

Most partial differential equations can not be solved analytically. For this reason, many numerical methods were developed to try and find appropriate approximated solutions to PDEs. The first method, based on the separation of variables on superposition of solutions of linear equations, was already studied in the 1740s by d'Alambert and Euler to solve the wave equation.

Many numerical methods are based on approximation of derivatives over a stancil

on a meshgrid. For time-dependent problems there can be two approaches: the method of lines and the method of lines transpose. The method of lines works by applying a spatial discretization to the domain of the problem while keeping time continuous. The result is a system of ordinary differential equations that can then be solved numerically with standard time integration schemes. The method of lines transpose, on the other hand, proceeds in the opposite order: a time discretization is performed first, using time-advancing schemes such as implicit or explicit Euler, yielding a sequence of time-independent spatial problems. In the case of parabolic equations, these problems are elliptic in nature and can be solved at each time step using standard spatial discretization techniques, such as finite element or finite difference methods.

2.2.1 Finite differences

Finite difference method is used to approximate derivatives using a discrete stencil on a meshgrid. Considering the mathematical definition of a derivative:

$$f'(x) = \lim_{h-0} \frac{f(x+h) - f(x)}{h}$$

and using Taylor expansion

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2} f''(x) + ... = f'(x) + O(h)$$

it is seen that an approximation for the first derivative

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

is accurate to first order.

Considering a grid with points at a distance Δx, and considering the values of a function on the grid points such that $f(i,j) = f(x_0 + i\Delta x, y_0 + i\Delta y)$, the first derivative of a function at a sampled point $x_i = x_0 + i\Delta x$ can be approximated by finite difference in different ways: using forward difference

$$\left(\frac{\partial f}{\partial x}\right)_{i,j} = \frac{f_{i+1,j} - f_{i,j}}{\Delta x} + O(\Delta x)$$

backward difference

$$\left(\frac{\partial f}{\partial x}\right)_{i,j} = \frac{f_{i,j} - f_{i-1,j}}{\Delta x} + O(\Delta x)$$

or central difference

$$\left(\frac{\partial f}{\partial x}\right)_{i,j} = \frac{f_{i+1,j} - f_{i-1,j}}{2\Delta x} + O((\Delta x)^2)$$

to obtain a more accurate approximation. The same thing can be done for second

derivative, where the resulting finite difference approximation will be:

$$\left(\frac{\partial^2 f}{\partial x^2}\right)_{i,j} = \frac{f_{i+1,j} - 2f_{i,j} + f_{i,j}}{(\Delta)^2} + O((\Delta x)^2)$$

2.2.2 Finite element methods

The finite element method is one of the most widely used methods in a multitude of applications. This method is more flexible than finite difference, since it does not rely on a grid but on a set of finite elements that can be built with different shapes and dimensions, and can thus be adapted to the characteristics of the domain and of the problem.

Considering the example of an elliptic PDE with an integrable source term and homogeneous Dirichlet boundary conditions:

$$\begin{cases} -\Delta u(x) = f(x), \quad x \in \Omega, \quad f \in \mathcal{L}^2(\Omega) \\ \qquad u(x) = 0 \quad on \quad \partial\Omega \end{cases} \tag{2.5}$$

we derive the weak formulation by integrating over the domain and multiplying both sides by a test function $v \in V$, where V is an appropriate function space:

$$-\int_\Omega \Delta u v = \int_\Omega f v$$

and integrating by parts we obtain:

$$\int_\Omega \nabla u \nabla v - \int_{\partial\Omega} \frac{\partial u}{\partial n} v = \int_\Omega f v$$

By choosing an appropriate set of trial functions such that they also respect boundary conditions, we obtain the weak formulation of the equation:

$$\int_\Omega \nabla u \nabla v = \int_\Omega f v \qquad \forall v = 0 \quad on \quad \partial\Omega, \qquad v \in V$$

Identifying the left-hand side as a bilinear form $a(u, v)$ and the right-hand side as a linear form $F(v)$, solving the elliptic equation means solving the variational problem of finding $u \in V$ such that $a(u, v) = F(v)$. The Lax-Milgram theorem guarantees the well-posedness of the problem under certain hypothesis on the continuity and coercivity of the forms.

Given a variational problem as above, the Galerkin method discretizes it by introducing a family of subspaces V_δ with a positive δ that tends to zero, of finite dimension N_δ, $V_\delta \subset V$ The discretized variational problem becomes

$$\begin{cases} find \quad u_\delta \in V_\delta \\ a(u_\delta, v_\delta) = F(v_\delta) \quad \forall v_\delta \in V_\delta \end{cases} \tag{2.6}$$

where V_δ is a Hilbert space, and F and a satisfy the same Lax-Milgram hypotesis.

Now, choosing a set of basis functions in V_δ

$$\phi_j, \qquad 1 \leq j \leq N_\delta$$

we can write the vectors as a linear combination of these

$$v_\delta = \sum_{j=1}^{N_\delta} v_j \phi_j$$

and the variational problem can be expressed as a set of N_δ equations by choosing each basis function as a test function:

$$a(u_\delta, \phi_j) = F(\phi_j) \qquad 1 \leq j \leq N_\delta$$

and representing u_δ with respect to the chosen basis functions

$$u_\delta = \sum_{k=1}^{N_\delta} u_k \phi_k$$

we obtain

$$\sum_{j=1}^{N_\delta} u_k a(\phi_k, \phi_j) = F(\phi_j) \qquad 1 \leq j \leq N_\delta$$

which is a linear algebraic system that can be written in matrix form as $A\mathbf{u} = \mathbf{f}$. The matrix of this algebraic system is symmetric and positive definite as a result of the Lax-Milgram hypothesis, and the system can hence be solved. This procedure can be easily extended to other partial differential equations with other boundary conditions.

The finite element method is typically formulated as a Galerkin method, in which the solution is approximated using basis functions with local support defined over a partition of the spatial domain. The domain is subdivided into simple geometric elements, that are most commonly triangles, in two dimensions, or tetrahedra, in three dimensions, though other shapes are also possible. These elements are required to satisfy certain regularity conditions to ensure stability, convergence, and good approximation properties of the method. In the Galerkin formulation, the solution is sought in a trial function space and the residual is tested against a test function space; in the standard formulation, these two spaces coincide. There exist many variants of the finite element method, which may differ in both the choice of elements and the definition of the trial and test function spaces. A relevant example is the Petrov-Galerkin method, in which the trial and test function spaces are chosen to be different. Another relevant class of methods is the discontinuous Galerkin methods, where the trial functions are allowed to be discontinuous across element interfaces and continuity is enforced through numerical fluxes.

The Finite element method is among the most widely used approaches for numerical solution of systems of partial differential equations, due to its flexibility and applicability across a broad range of problems. A key reason for its success lies

in its well-developed theoretical foundation: a comprehensive convergence theory, supported by rigorous a priori and a posteriori error estimates, provides strong guarantees on the stability, accuracy, and reliability of the method.

2.2.3 Finite volume method

The finite volume method is another method used to derive approximate solutions to PDEs. The main difference with the finite element methods is that FVM uses equations in their integral formulations, and is thus mostly used to solve PDEs that arise from physical conservation laws, since the integral form is the form in which equations are usually derived from physics. One of the fields in which it is most applied is fluid dynamics, as FVM naturally conserves the total flow of quantities in the cells.

If we consider an elliptic equation:

$$-\nabla \cdot (\mathbf{K}\nabla u) = f \qquad in \;\; \Omega$$

we can see that this standard form can be derived from the balance equation:

$$-\int_{\partial b} (\mathbf{K}\nabla u) \cdot \mathbf{n} ds = \int_b f dx \;, \qquad \forall b \subset \Omega$$

where $\mathbf{n}$ is the outward normal vector of ∂b. The standard form is derived from this using the balance equation for the conservation law:

$$\int_{\partial b} \mathbf{q} \cdot \mathbf{n} \; dS = \int_b f d\mathbf{x}$$

where $\mathbf{q}$ is the flux density, and the constitutive equation:

$$\mathbf{q} = -\mathbf{K}\nabla u$$

applying the Gauss theorem to the balance equation for the conservation law and substituting the constitutive equation we obtain the first formulation.

Once shown how the integral formulation is equivalent to the general form of the elliptic equation, it is possible to show how finite volume methods consider this integral formulation. Since FVM discretize the balance equation, the conservation holds throughout the discretization. The discretization approximates the function u by u_h in a space of dimension N_h, the domain by a finite subset $\mathcal{B} = b_i, i = 1 : M$ and the boundary flux on ∂b_i by $(\mathbf{K}\nabla_h u_h) \cdot \mathbf{n}$.

With these discretisations the finite volume method is to find $u_h \in V$ such that:

$$-\int_{\partial b_i} (\mathbf{K}\nabla_h u_h) \cdot \mathbf{n} \; dS = \int_{b_i} f d\mathbf{x} \;, \quad \forall \; b_i \subset \Omega, \quad i = 1 : M$$

The boundary flux of each element has to be assigned, and this can be done for example using finite difference. For an interior side shared by two elements it can be

defined by:

$$\nabla_h u_h \cdot \mathbf{n}_e := \frac{u_h|_{\tau_2} - u_h|_{\tau_1}}{c_{\tau_2} - c_{\tau_1}}$$

where τ_1 andτ_1 are the two elements sharing a side and $\mathbf{n}_e$ is the normal outward vector of the side in the element τ_1. The flux can also be discretized using finite element methods. In both these cases error bounds can be found using theory.

As in finite elements, there are many options of finite volumes that can be chosen and many variants, for example the Petrov-Galerkin formulation also exists for finite volume methods, where different trial space and test space are chosen.

2.2.4 VisualPDE

VisualPDE is an interactive solver for PDEs that is available online and runs on the device's browser [2]. It can be used to solve one dimensional and two dimensional systems of PDEs. It has been created with the objective to have a user friendly tool that allows to visualize the solution of PDEs making use of graphic processing units (GPUs).

The general set of equations that visualPDE is able to solve, with system of equation of maximum four unknowns u, v, w and q, is the following:

$$\begin{cases} \frac{\partial u}{\partial t} = \nabla \cdot (D_{uu}\nabla u + D_{uv}\nabla v + D_{uw}\nabla w + D_{uq}\nabla q) + f_u \\ \frac{\partial v}{\partial t} = \nabla \cdot (D_{vu}\nabla u + D_{vv}\nabla v + D_{vw}\nabla w + D_{vq}\nabla q) + f_v \quad or \\ v = \nabla \cdot (D_{vu}\nabla u + D_{vw}\nabla w + D_{vq}\nabla q) + f_v \\ \frac{\partial w}{\partial t} = \nabla \cdot (D_{wu}\nabla u + D_{wv}\nabla v + D_{ww}\nabla w + D_{wq}\nabla q) + f_w \quad or \\ w = \nabla \cdot (D_{wu}\nabla u + D_{ww}\nabla w + D_{wq}\nabla q) + f_w \\ \frac{\partial q}{\partial t} = \nabla \cdot (D_{qu}\nabla u + D_{qv}\nabla v + D_{qw}\nabla w + D_{qq}\nabla q) + f_q \quad or \\ q = \nabla \cdot (D_{qu}\nabla u + D_{qw}\nabla w + D_{qq}\nabla q) + f_q \end{cases} . \tag{2.7}$$

where each variable v, w and q can satisfy either a time-dependent PDE or can be expressed in terms of other variables and their derivatives. Diffusion coefficients are collected in a diffusivity matrix D and they can be functions of the unknowns, domain coordinates or any other parameter. f_i are given functions that can also depend on first and second spatial derivatives of $\mathbf{u}$. With this set of equations visualPDE is able to solve a very wide range of problems, including advection, cross diffusivity and highly nonlinear PDEs.

VisualPDE relies on traditional methods to numerically solve the systems. In particular, spatial derivatives are approximated with a finite difference scheme, and then the system is solved using a time-stepping scheme. The spatial domain considered is the width and height of the screen of the user's device, so it is mainly rectangular. Then we consider the coordinates (x, y) in $\Omega = [0, L_x] \times [0, L_y]$ for positive L_x, L_y. The user can define the spatial step size Δx and a $(L_x/\Delta x, L_y/\Delta x)$ grid is created on which spatial derivatives are computed using a finite difference scheme. First and second derivatives are approximated using a central difference scheme. VisualPDE

supports Dirichlet, Neumann, Robin, periodic or also mixed boundary conditions. Dirichlet BC are implemented by fixing the value of the unknown at the boundary, while other types of BC are enforced by adding ghost nodes to the grid and modifying the finite difference scheme. The ODE system resulting from the spatial discretization is solved by using an explicit, fixed-timestep finite difference scheme. The most used is the forward Euler scheme and the four step Runge-Kutta. RK4 is the most accurate one but also has the highest computational cost, while forward Euler is the simplest, but does not scale favourably with timestep.

VisualPDE exploits GPU capabilities to solve the large system of ODEs that arises from spatial discretization, and the key idea to do this is by casting the problem as an image manipulation, where the computational domain is represented by single floating point image textures and each pixel corresponds to a point in the spatial discretization. Points are updated in parallel every time step. The result is that visualPDE can update the solution many times each second.

Chapter 3

PINNs' State of the art

3.1 Neural networks

Artificial Neural Networks are computational models that replicate the structure of the neural system in biology. The first paper addressing Neural Networks is *The logical calculus of the ideas immanent in nervous activity* by Walter Pitts and Warren McCulloch, published in 1943 [3]. With their network they tried to replicate what happens in the brain, where neurons receive signals from other neurons through dendrites, process them and then produce an output that they fire to the next neuron through the axon and its ramifications. Further research regarding Artificial Neural Networks was brought forward in the 50s and 60s, but stagnated in the following years. New interest in this subject sparked in the 1980s after the backpropagation algorithm was popularized. Backpropagation uses the chain rule to compute gradients of the outputs with respect to the network's parameters, starting from the last layer and going backward to the input layer. The update of parameters is then computed according to the calculated gradients using a gradient-descent method.

A neural network is made up of several nodes (neurons) and edges (synapses). The nodes are organized in layers and are connected to each other through edges, each of which has a weight that multiplies the data transported to the next neuron. How these neurons and connections are organized can vary in each type of neural network. Different types of network arise from different connections. A layer is called fully-connected if each node receives input from every node of the previous layer. A further distinction concerns the directionality of connections. A network is called feed-forward when signal propagates strictly from one layer to the next, with no backward or lateral connections. A network is called recurrent when neurons can also receive inputs from neurons in the same or previous layers.

The structure of any neural network is made up of at least two layers: the input layer and the output layer. The input layer is not an active layer, since it does not perform any operation; it has as many neurons as the number of inputs of the problem considered, and its role is that of distributing inputs to the first active layer. The output layer is the last active layer and has as many neurons as the outputs of

the problem: the returned values are the values of the outputs.

The layers in between are called hidden layers and are made up of a variable number of neurons. Each neuron receives a weighted input from each neuron of the previous layer and computes the output by applying an activation function to the total input. The total input is the weighted sum of all the input plus a bias term. In a node i the output y_i is computed as:

$$y_i = \sigma(\sum_{j=1}^{N} x_j w_{i,j} + b_i)$$

where b_i is the bias pertaining to the i-th node, $w_{i,j}$ are the weights of the edges that connect node j to node i, x_j is the output of the node j and $\sigma(\cdot)$ is the activation function that acts on the summation and returns the output y_i.

Each output y_i will then be passed on to the next neurons through the edges and so on.

There are different functions that can be chosen as activation functions, and they should be chosen considering which one is more suitable for the problem based on their characteristics. Some of them are introduced in 3.1.2.

This type of feed-forward neural network computes an output given a set of inputs and a prescribed architecture. In order to produce the desired output, the network must be trained, meaning that an appropriate combination of weights and biases , that are called the learnable parameters of the network, must be found. Training is formulated as a minimization problem, where the objective is to find the set of parameters that minimizes a loss function measuring the discrepancy between the network's output and the target solution. The network is initialized with a set of parameters, which are iteratively updated throughout the training process. Each iteration consists of a forward pass, in which the input values are propagated through the layers of the network to produce an output. The loss is then computed, and the backpropagation algorithm determines the gradients of the loss with respect to all learnable parameters. These gradients are subsequently used by an optimization algorithm to update the weights and biases in the direction that reduces the loss. The choice of optimization algorithm depends on the problem at hand; different possible algorithms will be presented in Section 3.1.3.

The neural network operates in cycles, called epochs, each consisting of a forward pass, a loss evaluation, a backpropagation step, and a parameter update. Training continues until a stopping condition is met, which may be either that the loss falls below a prescribed threshold or that a fixed number of epochs, defined a priori, has been reached. The trained network is then able to return a sufficiently accurate approximation of the target output for any given input.

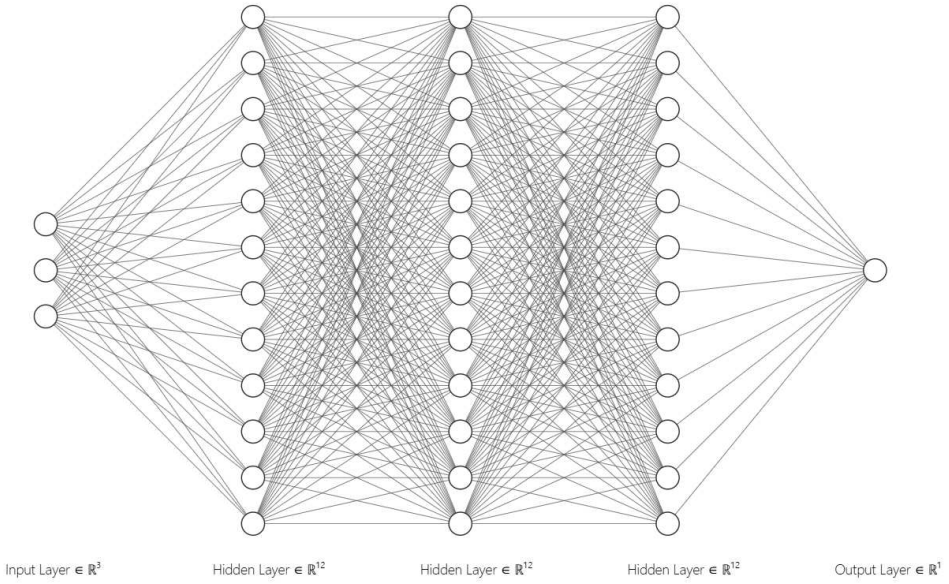

Figure 3.1: Scheme of a NN with three inputs, one output, three hidden layers of 12 neurons each

3.1.1 Depth and width

The terms *depth* and *width* refer, respectively, to the number of layers in the network and the number of neurons in each layer. Both have to be chosen carefully according to the specific problem. While increasing depth and width will result in increased expressivity, having too many learnable parameters in the network could lead to issues of overfitting. Furthermore, an excessive depth could lead to problems of vanishing gradients: since gradients are propagated backward through the chain rule, repeated multiplication by small values causes them to shrink exponentially in earlier layers. Also, computational cost scales with both depth and width. Figure 3.1 shows an example of a neural network made up of three hidden layers.

3.1.2 Activation functions

Activation functions are linear or non linear functions applied to the input sum:

$$I_i = w_{i1}x_1 + w_{i2}x_2 + .. + w_{in}x_n + b$$

where w_{ij} is the weight of the edge that connects node j to node i, and x_j is the output of the neuron j; b is the bias.

The application of the activation function $y_i = \sigma(I_i)$ results in the output of the neuron.

Neural networks employ non-linear activation functions to introduce nonlinearity into the model, enabling the approximation of complex, non-linear mappings. The choice of the activation function is crucial, as different functions exhibit distinct mathematical properties that significantly influence training dynamics, convergence

behaviour, and overall model performance.

The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid is a smooth and continuously differentiable function that maps real-valued inputs to the interval $(0, 1)$, and is commonly used especially in binary classification problems.

Another widely used activation function is the hyperbolic tangent function, of equation:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

The hyperbolic tangent maps real valued inputs to the interval $(-1, 1)$ and is centred around zero, which often results in more balanced gradient updates compared to the sigmoid function. It is smooth and differentiable, with steeper gradients near the origin.

The Rectified Linear Unit (ReLU) is defined as

$$f(x) = max(0, x)$$

This activation function introduces sparsity in the network by setting all negative inputs to zero, while leaving positive inputs unchanged. The advantage of ReLU is that it avoids the vanishing gradient issue and it has a low computational cost.

Each activation function has certain characteristics that are suitable for certain kinds of problem. The choice of the function can impact the efficacy of the network and the goodness of the results. For this reason, it has to be chosen according to the nature of the problem, together with empirical evaluation.

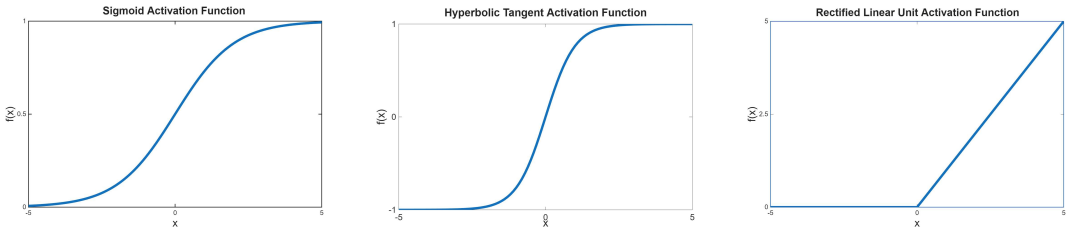

Figure 3.2: Activation functions

3.1.3 Optimization algorithms

Optimization algorithms update the network's learnable parameters in order to minimize the loss function. The choice of the optimizer can affect convergence properties and quality of the solution.

An extensively used algorithm is the Adam Update method, introduced in 2015 in [4], which is based on adaptive moment estimation. This algorithm adjusts the learning rate of the parameters θ and updates them using the first and second moment of the

gradients with the equation:

$$\theta_t = \theta_{t-1} - \frac{\alpha_t \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where α_t is the adaptive learning rate computed at each t as

$$\alpha_t = \frac{\alpha_{t-1}\sqrt{1-\beta_2^t}}{1-\beta_1^t}$$

and the first and second moments, $\hat{m}$ and $\hat{v}$ are updated with the following equations:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

where

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(g_t)^2$$

Here β_1, β_2 and ϵ are initialized hyperparameters. Adam method is often preferred to other methods because it has low memory requirements and it is suitable for noisy gradients. Furthermore, it is relatively insensible to the choice of the hyperparameters, making it a robust method in this sense.

Another algorithm is the Limited-memory Broyden-Fletcher-Goldfarb-Shanno, L-BFGS. It is a quasi-Newton optimization algorithm that uses information from previous gradients and parameter updates. This method has fast convergence for smooth optimization problems and performs well in problems that require high optimization precision.

Often, a combination of the two optimizers is used, finding a good approximation of the solution with Adam algorithm at first, and using L-BFGS to fine-tune the solution.

3.2 Physics-Informed Neural Networks

Neural Networks can be used in many fields of science as completely data-driven tools, but in the last decades the possibility of including underlying physics into the learning algorithm has been explored.

In 2017, Raissi et al. introduced Physics-Informed Neural Networks (PINNs) in their article *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations* [5]. PINNs are neural networks that integrate physical laws in the Deep Learning algorithm by including them in the loss function. The physical laws act as a constraint on the space of possible solutions, guiding the neural network to learn physically feasible solutions. Many physical constraints, such as conservation laws or symmetry, can be included by means of a PDE. This is a crucial advantage compared to purely data-driven machine learning,

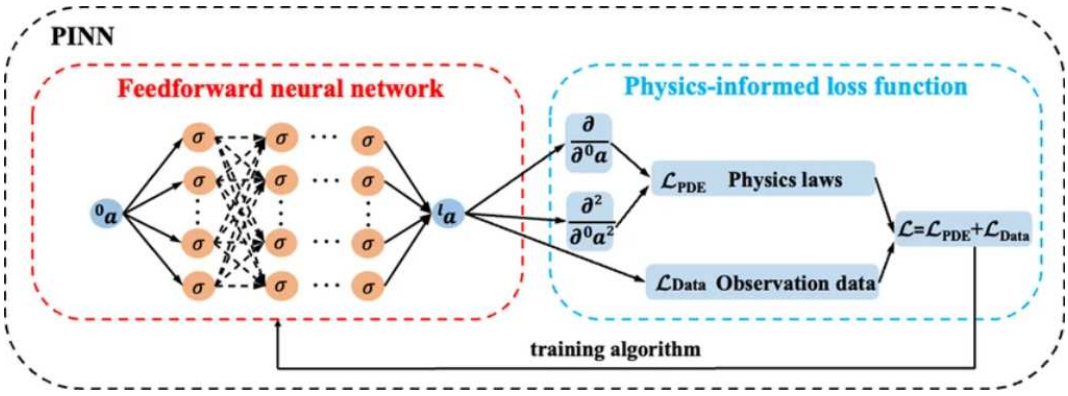

Figure 3.3: Scheme of a PINN - Scientific Figure on ResearchGate. Available from reference paper

especially in biological applications, since the collection of experimental data can in some instances be cumbersome.

PINNs can be used to infer equation parameters from a set of known data, or also as solver for PDEs, once initial and boundary conditions are known. This work focuses on the latter application.

To illustrate how PINNs work we take a generic Partial differential equation:

$$\mathbf{u}_t(\mathbf{x}, t) + \mathcal{N}[\mathbf{u}(\mathbf{x}, t)] = 0, \ \mathbf{x} \in \Omega, \ t \in [0, T]$$

where $\mathcal{N}$ is a differential operator, with some initial conditions

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{g}(\mathbf{x}), \ \mathbf{x} \in \Omega$$

and boundary conditions

$$\mathcal{B}[\mathbf{u}(\mathbf{x}, t)] = 0, \ \mathbf{x} \in \partial\Omega, \ t \in [0, T]$$

where $\mathcal{B}$ is an operator that can correspond to Dirichlet, Neumann, periodic or Robin boundary conditions.

A PINN is a neural network that takes as input the independent variables $\mathbf{x}$ and t and returns as output the value of the unknown function $u(\mathbf{x}, t)$. The key idea is to use automatic differentiation, which computes exact derivatives of the network output with respect to its inputs by traversing the computational graph, without requiring domain discretization. This allows the residual of the governing equation to be evaluated directly from the network output at any point in the spatio-temporal domain. The loss function consists of three contributions: the PDE residual, evaluated on a set of interior collocation points sampled from the spatio-temporal domain; the initial condition loss, computed on points drawn from the domain at t=0; and the boundary condition loss, evaluated on points sampled from the spatial boundary at random times. A scheme of PINN's architecture is illustrated in Fig. 3.3. The network is trained by minimizing this total loss, so that the physical law, the initial state, and the boundary conditions are simultaneously satisfied. Once

training is complete, the network returns an approximate solution at any point in the domain, without being restricted to the training points. This mesh-free nature is a key advantage over classical numerical methods such as finite elements or finite differences, which require a discretized domain and can become computationally expensive on complex or irregular geometries.

3.2.1 Loss function

The neural network returns an approximated solution $\mathbf{u}_{nn}(\mathbf{x}, t; \theta)$, obtained by minimizing a loss term $\mathcal{L}(\theta)$ on a set of collocation points, θ being the neural network trainable parameters. The total loss function is given by the sum of two terms: a data loss $\mathcal{L}_{data}$ and a physics loss $\mathcal{L}_{physics}$.

Data loss is only present when information on the value of the exact solution $\mathbf{u}$ on a certain set of spatio-temporal points $(\mathbf{x}_d, t_d)$ is available. Data is not necessary for the PINN to function, but, when available, it guides the network towards the correct solution. Data loss is defined as the squared error between the dataset of solutions and the current approximated solution evaluated on the same set:

$$\mathcal{L}_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} |\mathbf{u}_{nn}(\mathbf{x}_d^i, t_d^i, \theta) - \mathbf{u}(\mathbf{x}_d^i, t_d^i)|^2 \tag{3.1}$$

The physics loss term is where the physical constraints are enforced, and is given by the weighted sum of three different terms:

$$\mathcal{L}_{physics} = \omega_f \mathcal{L}_f + \omega_{ic} \mathcal{L}_{ic} + \omega_{bc} \mathcal{L}_{bc}$$

where ω_f, ω_{ic} and ω_{bc} are weights that have to be adjusted to balance the different terms.
The first term $\mathcal{L}_f$ is the square of the PDE residual

$$\mathcal{R}(\mathbf{x}_f, t_f, \theta) = \frac{\partial \mathbf{u}_{nn}(\mathbf{x}_f, t_f, \theta)}{\partial t} + \mathcal{N}[\mathbf{u}_{nn}(\mathbf{x}_f, t_f, \theta)]$$

evaluated on the N_f collocation points $(\mathbf{x}_f, t_f)$

$$\mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |\mathcal{R}(\mathbf{x}_f^i, t_f^i, \theta)|^2 \tag{3.2}$$

The other two terms are given by the squared difference between the approximated solution evaluated at initial and boundary points and the exact initial and boundary conditions respectively:

$$\mathcal{L}_{ic} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |\mathbf{u}_{nn}(\mathbf{x}_{ic}^i, t_{ic}^i, \theta) - \mathbf{u}(\mathbf{x}_{ic}^i, t_{ic}^i)|^2 \tag{3.3}$$

$$\mathcal{L}_{bc} = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |\mathbf{u}_{nn}(\mathbf{x}_{bc}^i, t_{bc}^i, \theta) - \mathbf{u}(\mathbf{x}_{bc}^i, t_{bc}^i)|^2 \tag{3.4}$$

where N_{ic} is a set of points in the spatial domain at time $t = 0$ and N_{bc} is a set of points chosen at different times on the boundary $\partial\Omega$. The total loss, that is the cost function for the minimization problem that the network has to solve, is the sum of all these terms:

$$\mathcal{L} = \mathcal{L}_{physics} + \mathcal{L}_{data} \tag{3.5}$$

3.2.2 Automatic differentiation

One of the main advantages of PINNs compared to traditional methods, as indicated before, is that they are mesh-free, and they are thus suitable to solve problems with complex geometries. This property stems directly from the use of automatic differentiation, which allows spatial and temporal derivatives to be computed exactly from the network output, without any domain discretization.

Automatic differentiation is a backpropagation method that exploits the chain rule to calculate derivatives of the output with respect to the parameters or the inputs of the network. In particular, in PINNs automatic differentiation is employed to evaluate the derivative of the output that are necessary to calculate the loss function. During the forward pass, a graph is created and values are stored in it at each step. Once the forward pass is over derivatives are obtained by consulting the traced graph and applying the chain rule at each node.

3.2.3 Collocation points

Collocation points are the points on which the residual is computed. They are usually randomly sampled from the spatio-temporal domain the PDE is defined on, but also other types of sampling can be chosen. For example, they can be uniformly sampled on an equispaced uniform grid, or using a Sobol sequence. Usually random sampling is preferred because it reduces computational cost and memory requirements, compared to full-batch sampling. In some cases the set of collocation points can also be chosen so that it improves the distribution of points during training: this practice is called adaptive re-sampling. Adaptive re-sampling aims at increasing the density of collocation points in areas where the loss, or the residual, has higher values in order to speed up convergence. Some examples are importance sampling [6] and residual-based adaptive refinement [7]

3.2.4 Notions on convergence theory

One important aspect about PINNs is the absence of a rigorous theory supporting their functioning, error estimates and convergence. In particular, there does not exist a unifying convergence theory that can be applied to general cases, but some results regarding convergence and error estimates have been derived for specific cases, under

specific hypothesis. In particular, many works focus on establishing negative results, highlighting the fundamental limitations of the method.

In Doumèche, Biau and Boyer [8] authors prove that unregularised PINNs can overfit, and provide a practical example for the heat equation, that can be generalized to other settings. They consider the limit of the empirical risk function, which is defined as:

$$R_{n_e,n_r}(u_\theta) = \frac{\lambda_e}{n_e}\sum_{j=1}^{n_e}|u_\theta(\mathbf{X}_j^e) - h(\mathbf{X}_j^e)|^2 + \frac{1}{n_r}\sum_{l=1}^{n_r}\mathcal{F}(u_\theta, \mathbf{X}_l^r)^2$$

where $\mathbf{X}^e$ are the n_e points for initial and boundary conditions, which are expressed by the condition $u(\mathbf{X}^e) = h(\mathbf{X}^e)$, and $\mathbf{X}^r$ are the collocation points in the domain where the residual is minimized; and the theoretical risk is expressed as:

$$\mathcal{R}(u) = \lambda_e\mathbb{E}|u(\mathbf{X}^e) - h(\mathbf{X}^e)|^2 + \frac{1}{|\Omega|^2}\int_\Omega \mathcal{F}(u, \mathbf{x})^2 d\mathbf{x}$$

They show that, for any pair of points and for any set of collocation and boundary points, there exists a minimizing sequence of approximate solutions u_θ such that the limit for the empirical risk function converges to zero but that of the theoretical risk function does not. This is an important result as it shows that a small loss value does not necessarily guarantee that the optimization error is low, and thus the loss is not a reliable parameter to determine whether the solution predicted by the network is accurate.

In Shin, Darbon and Karniadakis [9] authors focus on linear second-order parabolic and elliptic equations and show that for these equations a sequence of minimizers that satisfies initial and boundary conditions strongly converges to the solution in H^1 with respect to the number of training points.

For example, in [10] a priori and a posteriori estimates are provided for PINNs applied to linear PDEs, using coercivity and continuity. Furthermore, in the same work, results show that the norm of error decay is weakened by the vanilla approach, where initial and boundary conditions are softly enforced by minimization of the norm L^2. These two results suggest that, in general, the ability to strictly enforce initial conditions greatly enhances convergence of the method.

3.2.4.1 Errors

There are three types of errors that together make up the total error of PINNs: approximation error, optimization error and estimation error. The first one refers to the error that will occur between the real theoretical solution and the estimated solution due to the fact that the computed solution will be the output of an approximation model and not a true model. This error depends only on the architecture of the neural network. It has been showed that a wide-enough neural network can correctly approximate a wide category of functions and their derivatives, in particular neural network possess the so-called *universal approximation property* that means

that neural networks with at least one hidden layer are able to approximate any continuous function. The second type of error refers to the error between the obtained accuracy and the theoretical accuracy of the network. It is more difficult to estimate, since there are next to no guarantees that gradient-based optimization methods will be able to find the global minimum of the problem, in particular because the objective function of a neural network is highly non-convex, that means that it presents an important number of local minima and saddle points. The estimation error is the error in evaluating the model at unknown points. This, together with approximation error, make up what is called generalization error, that means the distance between the global minimizer of the loss and the real solution to the PDE that we want to approximate. In general, many studies on PINNs error focus on obtaining bounds for the generalization error, neglecting the contribution of optimization error, even though the term can be quite relevant since often PINNs are not able to reach the global minimum of the loss function.

3.2.4.2 Neural Tangent Kernel framework

An interesting and important part of convergence theory for PINNs relies on what is called the Neural Tangent Kernel theory. This framework explores the connection between deep neural networks and kernel regression methods, providing insight into the dynamics of gradient-based training.

The neural tangent kernel is a kernel that captures the dynamics of a fully-connected network trained by gradient-descent algorithm in its infinite width limit. Considering a fully connected neural network of function $f(\mathbf{x}, \theta)$, where θ are the net learnables and $\mathbf{x}$ are the input data, that minimizes the mean square loss $\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} |f(\mathbf{x}_i, \theta) - y_i|^2$, its neural tangent kernel operator $\mathbf{K}$ is defined as follows:

$$\mathbf{K}_{i,j} = \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \frac{\partial f(\mathbf{x}_i, \theta)}{\partial \theta}, \frac{\partial f(\mathbf{x}_j, \theta)}{\partial \theta} \right\rangle$$

In the infinite-width limit, it can be shown that the network output at initialization converges to a Gaussian distribution. Moreover, if the network uses an infinitesimally small learning rate and a gradient descent algorithm, the kernel $\mathbf{K}$ converges to a fixed kernel $\mathbf{K}^*$ that is constant throughout training, since it depends only on the depth of the network, the variance of the initialisation distribution and the choice of nonlinearity. This kernel matrix is important because its positive-definiteness is strictly related to the convergence properties of the network, it is shown that convergence is faster along the directions corresponding to the largest eigenvalues, which in practice correspond to lower-frequency components of the solution. This gives rise to the phenomenon known as *spectral bias*, meaning that the network preferentially learns low-frequency features before high-frequency ones. The spectrum of the NTK at initialization therefore determines the convergence rate of the total error during training, and provides a theoretical justification for the difficulties PINNs encounter when approximating multi-scale solutions. Furthermore, in Wang, Yu and Perdikaris [11] authors use the NTK framework to show that there exists a

discrepancy in the order of magnitude of the different types of losses. In fact, while building a PINN it is important to select appropriate weights for each term in the loss function. PINNs are very sensitive to the balance between the different losses and a wrong choice could result in convergence to a local minimum.

3.2.5 Challenges

PINNs have proved to be very effective in data-driven discovery of parameters and in data-driven discovery of solutions, while they encounter more struggles when they are employed as PDE solvers without using any data to drive them to the right solutions. This section will focus on this last application, analysing the challenges PINNs face when being employed as PDE solvers.

Spatial domain

PINNs have been seen to be struggling in particular with big spatio-temporal domains and with stiff problems that present discontinuities. In order for PINNs to be able to capture the dynamics of a PDE in a big domain it is necessary to select a great number of collocation points. The increase in collocation points impacts computational cost, causing the convergence to be very slow and thus preventing a correct approximation of the solution.

A possible solution to this problem is that of performing a domain decomposition. In [12] XPINNs (Extended Physics-informed Neural Network) is proposed. The framework is based on decomposing the space-time domain into subdomains, in each of which a neural network is trained separately, with the possibility to choose selected parameters and architectures. On each interface a continuity condition is imposed on the residual, along with the average solution on the two subdomains sharing an interface. XPINNs enable parallelization, since in each subdivision a different neural network is trained, and an efficient adaptation of parameters that suit each zone of the domain.

Another proposed framework is the Adversarial and Self-Adaptive Domain Decomposition PINNs (AS-PINNs). In [13] a method is introduced, that automatically adjusts the positions of interfaces between subdomains in order to capture discontinuities without the need of manual handling of boundary location, and without needing any prior information on the location of the discontinuities. AS-PINNs employ adversarial method to minimize the loss function by automatically adjusting the boundaries to locate discontinuities.

Time domain

Long-time simulations cause PINNs to struggle. Since the collocation points are in a spatio-temporal domain, longer simulations require a greater number of collocation points. Furthermore, since the residual is minimized simultaneously at all points, the presence of very long times can violate causality. To overcome this problem, it is

possible to train PINNs in time slabs, where each slab takes as initial condition the values obtained by the previous network. This practice helps accuracy but greatly increases the computational cost, since many networks need to be trained, and they cannot be parallelized.

Mini batching

Mini-batching is a technique that consists in sampling random points from the set of all collocation points at each iteration and calculating loss only on the selected subset. Implementing mini-batching results in decreased memory requirements and faster optimization since the number of collocation points on which the algorithm works in each epoch is reduced.

Loss balancing

PINNs have to deal with different scales of losses, since the different terms can have different order of magnitude. Each loss is thus multiplied by a weight that aims at balancing the different terms. The easiest way to define the weights is to choose them manually prior to the training, but optimal weights are very problem specific and it can be difficult to determine which is the best combination of weights for each case. It is possible to use a self-adaptive algorithm that updates the weights during training, ensuring that the norms of the gradients of the weighted losses are of the same order. This algorithm tackles the problem of balancing initial, boundary, and residual loss. However, residuals and IBC losses can still differ severely from one collocation point to another, resulting in an unbalanced convergence of losses. One technique that can be employed to tackle the problem is self adaptive PINNs (SA-PINNs), as introduced in [14]. In SA-PINNs every collocation point is assigned a weight that can be updated during training. Weights are increased where the loss on the corresponding training point is increasing, this is done by including the weights as a trainable parameter. In this variation of PINNs the network is solving a min max problem, minimizing the losses and maximizing the weights, and this corresponds to finding a saddle point in the cost surface. Including self-adaptive weights has been experimentally proven to be effective in bettering the accuracy in solving stiff problems, with which classic PINNs struggle. Furthermore, in [14] the Neural Tangent Kernel matrix of SA-PINNs has been derived and it has been shown that the distribution of the eigenvalues is evened out compared to classic PINNs.

Stiff problems and spectral bias

It has been proved that PINNs suffer from spectral bias, meaning that they struggle to capture high frequency behaviours of the solution. This phenomenon has been observed in several practical applications and also been explained using the Neural Tangent Kernel theory.

To enhance the ability of PINNs to learn higher-frequency solutions, it is possible to express the input via a Fourier feature mapping. A Fourier feature mapping

γ performs a transformation on each input coordinate using a series of sinusoidal functions of different frequencies and it is written as:

$$\gamma(\mathbf{v}) = \begin{bmatrix} cos(\mathbf{Bv}) \\ sin(\mathbf{Bv}) \end{bmatrix}$$

with σ a positive hyper-parameter, $\mathbf{B}$ a $m \times d$ matrix with real-valued entries sampled from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

The mapping is used as a coordinate embedding of the network's input and it can be viewed as a layer with no trainable parameters. The value of σ determines which is the preferred frequency solution for the network to learn. It is also possible to create two separate embeddings, one for space coordinates and one for time coordinates, since there could exist multi-scale behaviours also in time. The employment of Fourier Feature embedding into the PINN algorithm is proved to lead to more accurate result, and to enhance the capability of the network to learn higher-frequency solutions in cases where normal PINNs struggle to. It is significant to note that the embedding does not add many more floating-point operations, so it does not impact the overall computational cost. In order to have an effective embedding, the frequency of the leading eigenvector of the neural tangent kernel has to agree with the frequency of the target function. This is possible only if there exists a priori knowledge on the eigenvectors, which is usually not available. In this case, a suitable value of σ has to be chosen with a trial and error approach, keeping in mind that values too large may lead to overfitting, while choosing too small values could fail to solve the spectral bias problem. Overall, Fourier Feature embeddings are a promising approach to tackle the problem of PINNs' spectral bias, but the practicality of their enforcing might not always be easy.

Lack of causality, propagation of information from initial condition to later times

Due to their structure, that is based on evaluating the solution simultaneously in a certain number of spatio-temporal points randomly allocated in the domain, PINNs are not compelled to respect causality. Furthermore, using the NTK framework it can be seen that PINNs are biased towards first minimizing the loss at later times. This suggests that causality can and will most probably be violated while learning the solution. In time-evolving problems this could strongly impact the physical accuracy of the solution. In Wang, Sankaran and Perdikaris [15] the authors propose a way to tackle this problem by creating a loss function that does not evaluate the loss at higher time points unless the loss of points at lower times is low enough. This ensures that when the approximate solution is computed at a certain time the solution corresponding to previous times is accurate at least up to a certain point. This causal training does not add a significant computational cost.

Overfitting

As stated before, PINNs are subject to overfitting, since there could exist solutions that minimize the residual on the collocation points, and can thus produce a very small value of the loss, but still do not provide a good approximation of the real function in other points of the domain. When this happens, it is usually related to a large value of the parameters θ of the network. A proposed regularization strategy is thus that of adding a term to the total loss that prevents the parameters values from becoming too large

$$\mathcal{L}_{tot} = \mathcal{L}_{IC} + \mathcal{L}_{BC} + \mathcal{L}_f + ||\theta||_2^2$$

3.2.6 Vanilla PINNs and Hard PINNs

The vanilla PINN formulation enforces initial and boundary conditions by including in the loss function a penalty term that measures the L^2 norm of the discrepancy between the network output and the prescribed conditions at a set of boundary and initial points. Ideally, the network will be guided towards minimizing the loss on the boundaries, thus respecting the imposed conditions, but in many cases the balancing of different loss terms is difficult, and initial and boundary conditions are not respected. In particular,in multi-scale problems the presence of a wide range of scales in the dynamics might cause certain loss terms to dominate others,leading to incorrect approximations. In some classes of problems, such as stiff reaction-diffusion equations, the temporal evolution of the solution is highly sensitive to the initial condition. The lack of a strict enforcement of initial and boundary conditions in vanilla PINNs hinders the capability of the network to capture physically correct behaviours in time, as any error in the initial state propagates and corrupts the solution throughout the entire spatio-temporal domain.

Theoretical results

The theoretical consequences of soft constraint enforcement have been studied in several works. In [10] a priori and a posteriori error estimates are derived and it is shown that the L^2 penalty approach for initial and boundary conditions reduces the rate of error decay compared to an exact enforcement. In particular in [16] the authors show that H^2 approximation and a successful training result in convergence in $H^{1/2}$ for vanilla PINNs. In contrast, in [17] it is shown that for ansatz classes that satisfy boundary conditions it is possible to exactly yield error estimates in H^2. These results have been provided for linear elliptic and parabolic equations. While they cannot be extended to arbitrary PDEs, they provide strong theoretical motivation for the development of hard-constrained PINN formulations, in which conditions are embedded directly into the architecture rather than penalized in the loss.

Hard PINNs implementation

Several methods have been proposed in the literature to strictly enforce initial and boundary conditions in PINNs . The simplest approach consists in increasing the weights assigned to the initial and boundary loss in the total loss function. While this can improve the solution in some cases, it does not ultimately resolve the problem, since the conditions are still not satisfied exactly. In stiff problems in particular, certain loss components tend to dominate others due to the wide range of scales present in the solution, making it necessary to perform a careful, problem-specific tuning of the loss weights [18].

Since neural networks are not designed for constrained optimization, it is necessary to encode constraints directly into the architecture. To do this, the unconstrained neural network has to be transformed into an ansatz space that respects the desired boundary conditions.

Before describing the available methods, it is useful to recall the main types of boundary conditions for PDEs.

Dirichlet boundary conditions, also called essential boundary conditions, give the solution's value on the boundary:

$$\mathcal{B}(u)(t, \mathbf{x} =)u(t, \mathbf{x})$$

Neumann boundary conditions give the value of the solution's normal derivative on the boundary, and are thus more tricky to impose as hard constraints:

$$\mathcal{B}(u)(t, \mathbf{x}) = \frac{\partial u}{\partial \mathbf{n}}(t, \mathbf{x})$$

where $\mathbf{n}$ is the outward normal vector to $\partial\Omega$. Combining these two, Robin boundary conditions are obtained:

$$\mathcal{B}(u)(t, \mathbf{x}; a, b) = au(t, \mathbf{x}) + b\frac{\partial u}{\partial \mathbf{n}}(t, \mathbf{x})$$

Periodic boundary conditions are made of d equations:

$$\mathcal{B}(u)(t, \mathbf{x}) = u(t, \mathbf{x}) - u(t, \mathbf{x} + P\mathbf{e}_i) = 0, i = 1.....d$$

where P is the period and $\mathbf{e}_i$ is the base vector in $\mathcal{R}^d$.

One of the earliest approaches to hard constraint enforcement was proposed by Lagaris et al. (1998), where the trial function is decomposed into an analytic term that satisfies the boundary conditions exactly and a second term given by the product of the neural network output and a function that vanishes on the boundary.

For Dirichlet boundary conditions, Nitsche's method [19] offers a variational approach to their enforcement.

Another method is the hPINNs, introduced in [20] .hPINN uses two different methods: the penalty method and the augmented Lagrangian method. The first

one works by replacing the constrained optimization problem with a sequence of unconstrained problems. Each problem has a coefficient that is increased by a constant factor at each iteration. After the first iteration, where the networks are trained from a random initialization, the subsequent ones are trained starting from the solution of the previous one. In this way, the solution of the unconstrained sequence converges to the solution of the original constrained problem. The second method is also a method that uses penalty terms but additionally incorporates terms that imitate the Lagrange multipliers. This latter method has a better convergence rate compared to the penalty method.

For Neumann boundary conditions a method that has been proposed in [21]is that of using carefully chosen Fourier feature embeddings. The idea is to restrict the network to use only cosine features with integer frequencies, which are flat at the spatial boundary, and to augment the output with a correction term to account for non-vanishing Neumann data. This approach is reported to be more computationally efficient and easier to implement than existing alternatives.

The most widely used approach for enforcing initial and boundary conditions exactly is based on the use of distance functions [22]. The idea is to apply the following transformation to the approximate solution u_{nn} of the neural network:

$$\tilde{u}(t,x) = \psi(t,x) + \phi(t,x)u_{nn}(t,x)$$

where ψ and *phi* are smooth functions and have the following properties:

$$\phi(0,\mathbf{x}) = u_0(\mathbf{x}),\ \psi(0,\mathbf{x}) = 0,\ for\ x \in \overline{\Omega}$$

for periodic boundary conditions it is required for both the functions to be periodic, while for other boundary conditions they will have to satisfy the conditions:

$$\phi(t,\mathbf{x}) = g(t,\mathbf{x}), \psi(t,\mathbf{x}) = 0\ for\ (t,\mathbf{x}) \in [0,T] \times \partial\Omega$$

so that the new u is equal to the initial condition at time zero and is always equal to the boundary conditions on the boundary.

The transformed u is the one that has to be considered in the computation of the physics loss, and the initial and boundary losses are comprised in this term.

The loss term becomes:

$$\mathcal{L}(u_{nn}) = \frac{1}{N_{CL}} \sum_{i=1}^{N_{CL}} |(\tilde{u} - \mathcal{P}(\tilde{u})(t_i^{Cl}, \mathbf{x}_i^{CL})|^2,\quad for\ \ \tilde{u} = \psi + \phi u_{nn}$$

With this method the difficulty of the computation of the loss has slightly increased but the initial and boundary conditions are enforced exactly, so the corresponding loss terms do not need to be considered. This approach has been shown to improve accuracy and training efficiency of PINNs.

Chapter 4

Application and results

In this chapter we present two case studies to analyse the performance of PINNs. The first application is the solution of the one-dimensional Burgers' equation. The solution to the equation will be approximated using a PINN built in MATLAB, and the result will be compared to the analytical solution.

The second application will be to the Gray-Scott system in a two dimensional space domain. An approximation of the solution will be computed using both classical numerical methods and a PINN. The goal of this application is to compare the efficacy of PINNs to that of traditional numerical methods. Four different solvers will be used to approximate the solution: firstly, we will propose a visualization of the solution using the website VisualPDE, that allows us to see the patterns that arise from the time evolving system; secondly, we will solve the equation using the two most widely used classical solvers: a finite element method, and a finite volume method. Lastly, both a Vanilla PINN and a hard-constrained PINN will be used, to verify the capability and efficacy of this method and compare the two different formulations.

4.1 Background

4.1.1 Burgers' equation

Burgers' equation is a non-linear parabolic equation that was first studied by Bateman and later by Burgers, to whom it owes the name. It often occurs in different areas of science, such as fluid mechanics and mathematical physics, as well as in biology to model shock waves and transport of a species. The equation is:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = \nu \nabla^2 \mathbf{u}$$

where $\mathbf{u}$ is the velocity field and ν is the diffusion coefficient, that must be a positive value.

The equation is an advection-diffusion equation; the advection term is the non linear term on the left-hand side, and the right-hand side contains a linear diffusion term.

The equation describes a quantity that travels through advection and diffuses

simultaneously, and the solutions that are generated will have sharp gradients, as an effect of the non linear advection term. Sharpness is smoothed out over time as a result of diffusivity. Analytical solutions can be calculated for the one-dimensional equation

$$u_t + uu_x = \nu u_{xx} \tag{4.1}$$

using the Cole-Hopf transformation. This transformation linearizes the equation and the analytical solution that is found is used as a relevant benchmark to validate numerical schemes.

4.1.2 Gray-Scott equation

The Gray-Scott equation is part of the class of reaction-diffusion equations. A generic equation in this class is of the form:

$$\frac{\partial \mathbf{u}(\mathbf{x}, t)}{\partial t} = D\nabla^2 \mathbf{u}(\mathbf{x}, t) + \mathbf{f}(\mathbf{u}(\mathbf{x}, t))$$

where $\mathbf{u}$ is a vector of chemical factors that interact, D is the diffusivity matrix, which will be a diagonal matrix with strictly positive entries, and $\mathbf{f}$ defines the chemical reactions. These kinds of equations, under a set of conditions on the reaction functions and on the diffusivity coefficients, give rise to the so called diffusion driven instability. Diffusion driven instability is the phenomenon by which diffusion destabilises a system that would otherwise remain stable in the absence of spatial variations, and the instability produced leads to spatial reorganization; that is what causes pattern formation.

The Gray-Scott equation describes the chemical interaction between two species U and V

$$\begin{aligned} U + 2V &\to 3V \\ V &\to P \end{aligned} \tag{4.2}$$

where V replicates, while consuming U, and V decays turning into an inert product P at a steady rate.

The planar Gray-Scott equation is:

$$\begin{cases} \frac{\partial u}{\partial t} = D_u \nabla^2 u - uv^2 + F(1-u) \\ \frac{\partial v}{\partial t} = D_v \nabla^2 v + uv^2 - (F+k)v \end{cases} \tag{4.3}$$

where u and v are the chemicals' concentrations, F is the feed rate at which the chemical U is replenished, and k is the kill rate at which V is removed, D_u and D_v are the diffusion coefficients.

The values of the parameters directly influence which kind of pattern will develop in time. Different combinations can produce stripes, spots, circles and labyrinth-like shapes.

The system admits three steady states: one is the trivial equilibrium at $u = 1$,

$v = 0$, while the other two are non-trivial and exist only when F and k satisfy certain conditions. Of these, one is always a saddle point, while the other can change stability depending on the values of the parameters. It is this latter equilibrium that undergoes a Turing instability, breaking spatial homogeneity and giving rise to the variety of patterns that the Gray-Scott model is known for .

4.2 Case study 1: Burgers' equation

The first case study is the solution of a one-dimensional Burgers' equation using a PINN. The approximation provided by the PINN will be compared to the analytical solution of the equation, calculating the loss and the error, to provide proof of how PINNs are a successful meshless tool in solving this kind of equations.

4.2.1 Experimental setup

The one-dimensional Burgers' equation (4.1) is solved with a viscosity value of $\nu = \frac{0.01}{\pi}$ in the domain $(x,t) \in (-1,1) \times (0,1)$, with the following initial and boundary conditions:

$$\begin{cases} u(x,0) & = -sin(\pi x) \\ u(-1,t) & = 0 \\ u(1,t) & = 0 \end{cases}$$

The approximated solution to the equation is computed using a Physics-Informed Neural Network implemented in MATLAB. The implementation relies on the use of the MATLAB Deep Learning Toolbox, that allows the computation of derivatives using automatic differentiation. The architecture is made up of 8 layers, each of which has 20 neurons with hyperbolic tangent activation functions. The solution is computed on a set of 10000 collocation points randomly sampled from the spatio temporal domain, and the initial and boundary conditions are enforced on 50 points each, also randomly sampled from the boundary. The network is trained iteratively for 1500 epochs with the L-BFGS optimization algorithm.

4.2.2 Results

The simulation was run on a medium-power laptop and it took 695.8 seconds to complete, of which 694.8 seconds was the time needed to complete the training of the network and 1 second the time used to evaluate the solution at the requested times and points. Figure 4.1 shows the results of the simulation at different times: the red dotted line is the analytical solution of the equation, and the blue continuous line is the solution predicted by the PINN. It can be seen that the two overlap almost completely in all of the time steps, showing that the PINN was able to produce a good approximation of the exact solution of the one-dimensional Burgers' equation consistently in the whole domain. Figure 4.2 plots the absolute value of the error between the analytical solution and the approximation obtained with the PINN. The

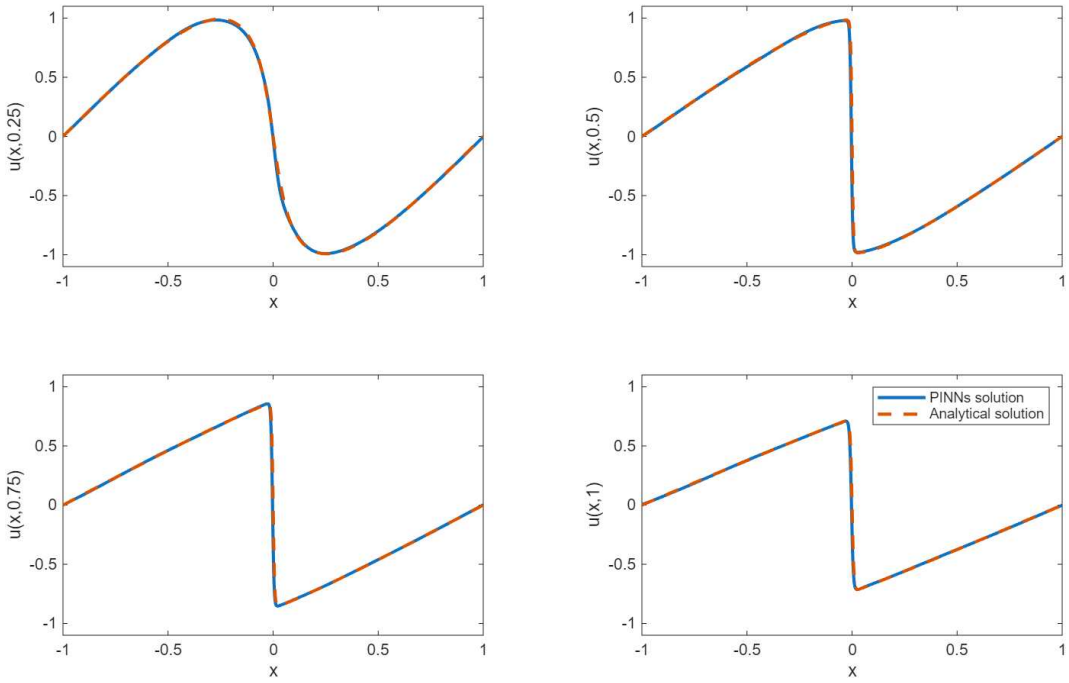

Figure 4.1: Comparison of PINN's solution to 1D Burgers' equation with the analytical solution

plots reveal that the highest values for the error, at each time, are lumped around the shock area, where the gradient of the function is sharper. This result is coherent with the expected outcome, since PINNs struggle to approximate sharp gradients functions

Figure 4.3 represents the loss function. It can be seen that loss decreases rapidly in the fist epochs and then flattens out, reaching a value of 1.639 after approximately 1000 iterations, stabilizing for the remaining number of epochs.

Overall, the results demonstrate a satisfactory performance of the PINN framework in this application. The relatively small spatial domain and the moderate smoothness of the solution, combined with a well-conditioned loss landscape, provide a favourable setting for the employment of PINNs. Nevertheless, the localised increase in error near the shock front highlights a known limitation of the method.

4.3 Case study 2: Gray-Scott system

4.3.1 Experimental setup

We consider the two-dimensional Gray-Scott system of equations (4.3), with the following parameters:

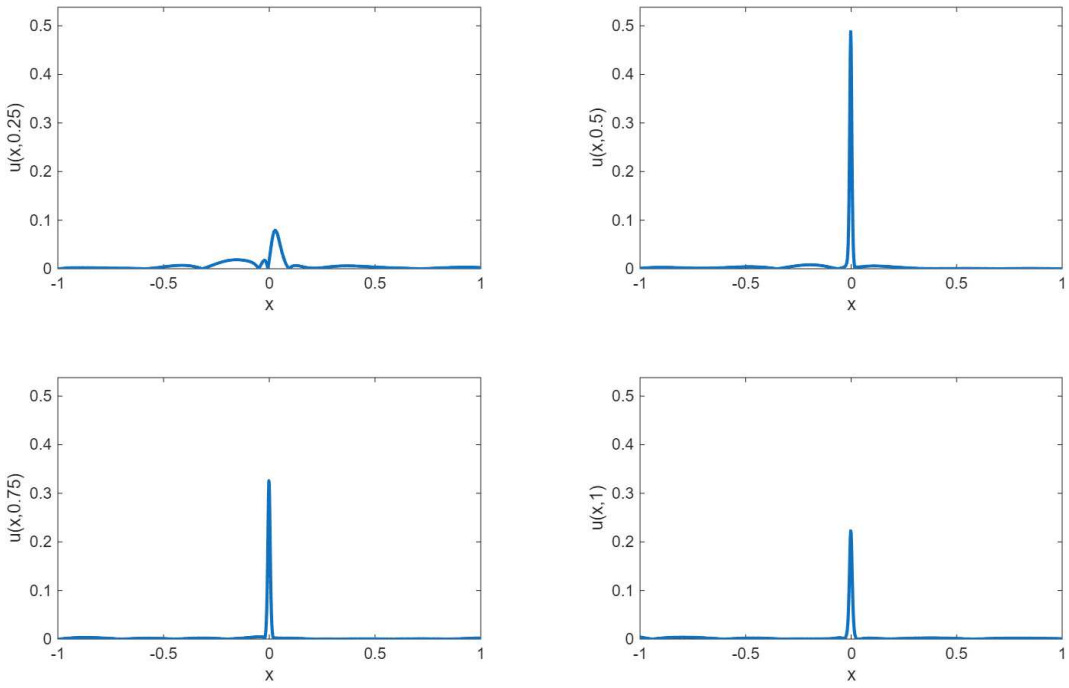

Figure 4.2: Absolute value of the error between the predicted and analytical solution

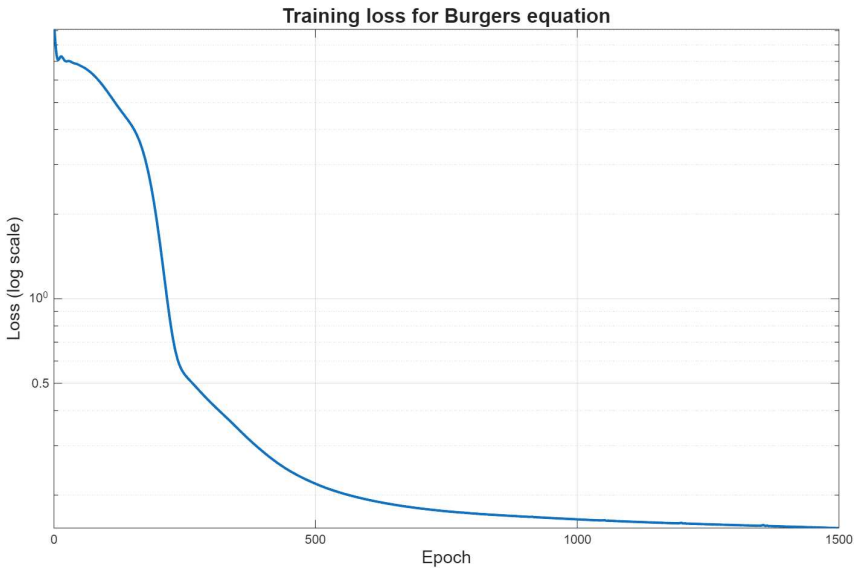

Figure 4.3: Loss value at each epoch for the solution of the 1D Burgers' equation

F	0.037
k	0.06
D_u	2
D_v	1

Table 4.1: Parameters for the two-dimensional Gray-Scott system

The equations are set in a two-dimensional domain in the plane (x, y) of dimension $\Omega = (-200, 200) \times (-200, 200)$, and the time domain is $t \in (0, 2000]$. The boundary condition is set to be a zero-flux Neumann boundary condition on all of the domain. The chosen initial condition is:

$$u(x, y, 0) = 1 \quad for \ \ \mathbf{x} \in \Omega \tag{4.4}$$

$$v(x, y, 0) = \begin{cases} 1 & if \ \ x^2 + y^2 \leq 16 \\ 0 & otherwise \end{cases} \tag{4.5}$$

4.3.2 Results and discussion

We present the solution to the Gray-Scott system (4.3) obtained using three different classical methods. The first solution is a visualization of the patterns obtained using VisualPDE. Initial conditions are applied with a "brush" function, that was set to impose a value of one in a circle of radius $r = 4$. In visualPDE the domain is automatically set as the dimension of the user's monitor. The time-stepping scheme was chosen to be the forward Euler, with a time-step of dimension 1.

Figure 4.4 and 4.5 illustrate the results at times 0, to show the initial condition that is given, at time t=100 and t= 1000, where it is possible to see the initial propagation and formation of a pattern, and t= 2000 to visualize the actual evolution of the pattern. The results show that VisualPDE is an effective solver for these kinds of problems, managing to capture the difficulty of the patterns.

Then the system is solved using a solver in MATLAB, that relies on the use of a Finite Element scheme. The MATLAB solver uses the Partial Differential Equation toolbox. The toolbox provides a built-in function that takes as input the model of the PDE and the solution times requested, and returns the solution evaluated at the given times. The PDE model defines the parameters as well as the initial and boundary conditions. The simulation took 71 seconds to complete on a medium-power laptop. Figures 4.6 and 4.7 show the results obtained with this method for the species u and v respectively, evaluated at the same times as the ones above. The method correctly enforces the initial condition of a disk of radius 4 and value one in the centre of the domain. Furthermore, comparing the figures to Fig. 4.4 and Fig. 4.5 we can see that both the shapes and the values of the chemicals evolve with the same speed in the two solvers, and the final pattern is the same.

The third solver is the Finite Volume Method, implemented in Julia. As with the previous methods, the results demonstrate that this approach successfully captures

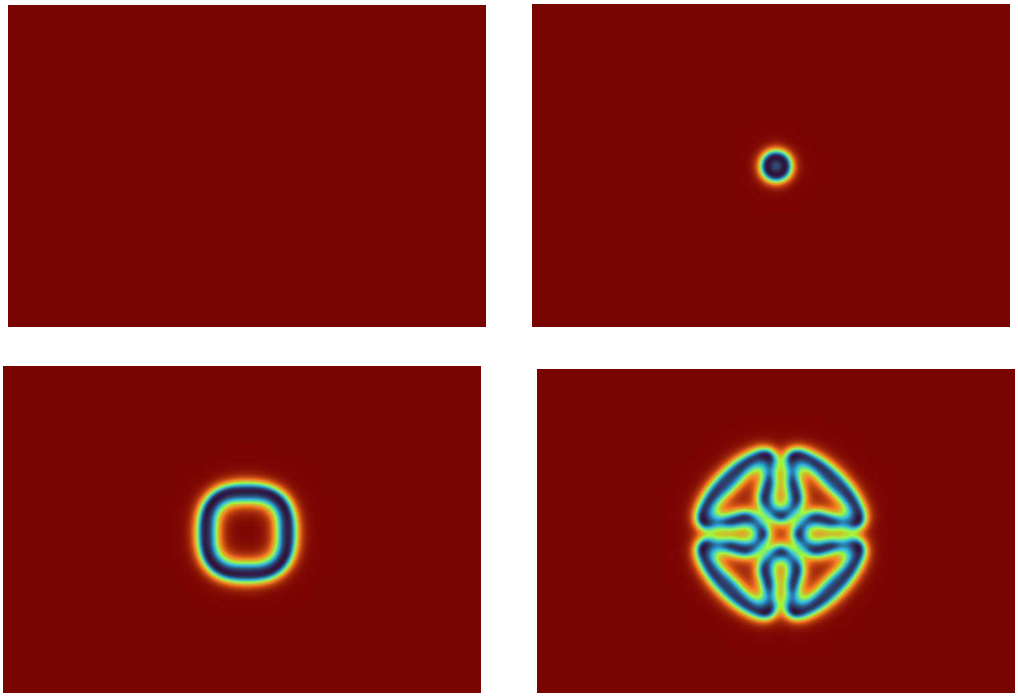

Figure 4.4: Solution of the 2D Gray-Scott system: Values of u at times $t = 0$, $t = 100$, $t = 1000$ and $t = 2000$ with VisualPDE

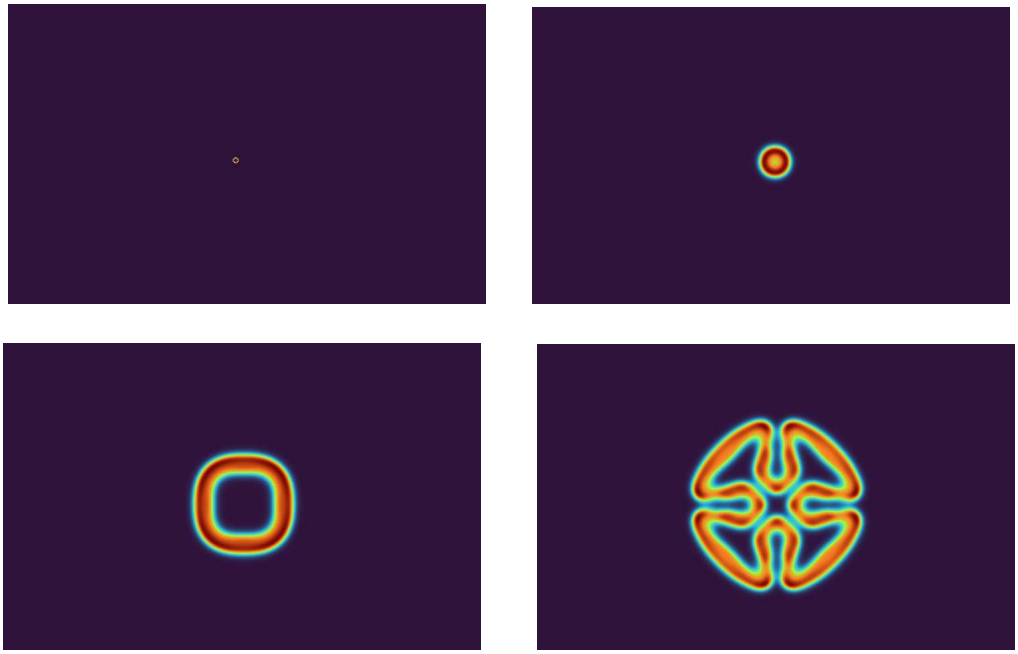

Figure 4.5: Solution of the 2D Gray-Scott system: Values of v at times $t = 0$, $t = 100$, $t = 1000$ and $t = 2000$ with VisualPDE

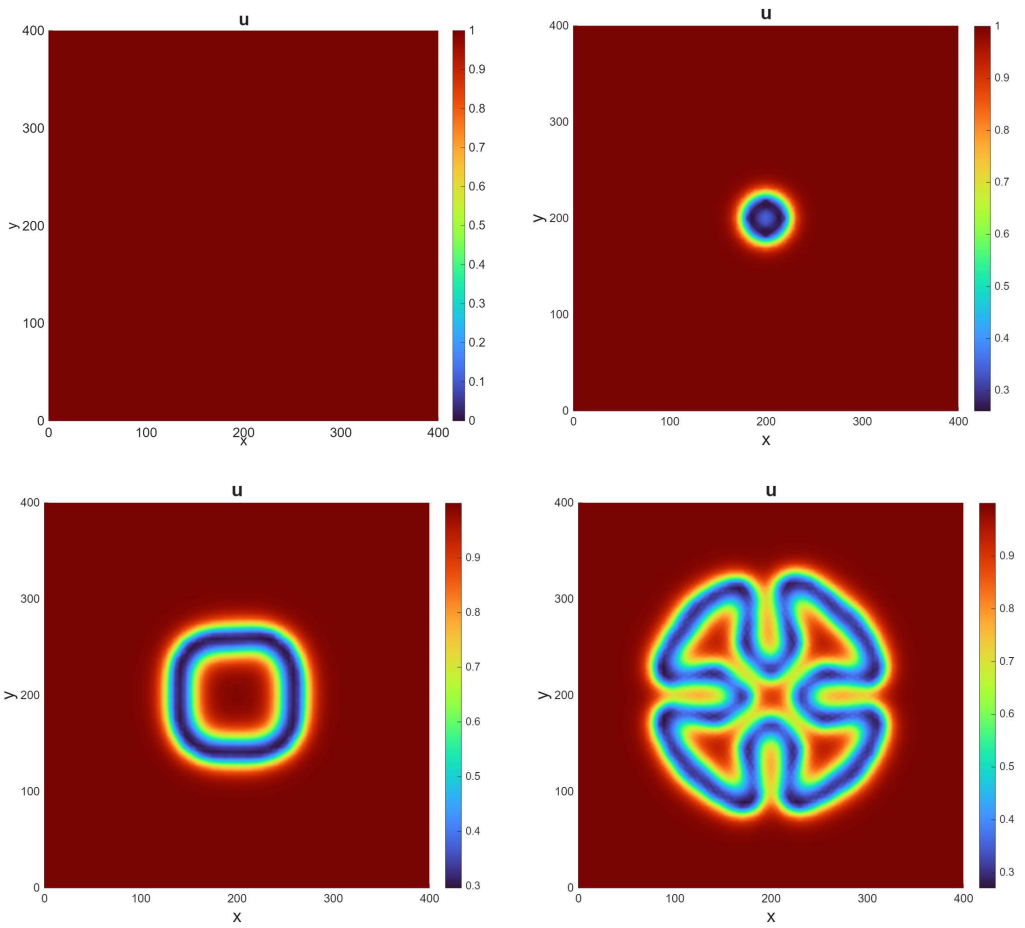

Figure 4.6: Solution of the 2D Gray-Scott system: Values of u at times $t = 0$, $t = 100$, $t = 1000$ and $t = 2000$ with FEM

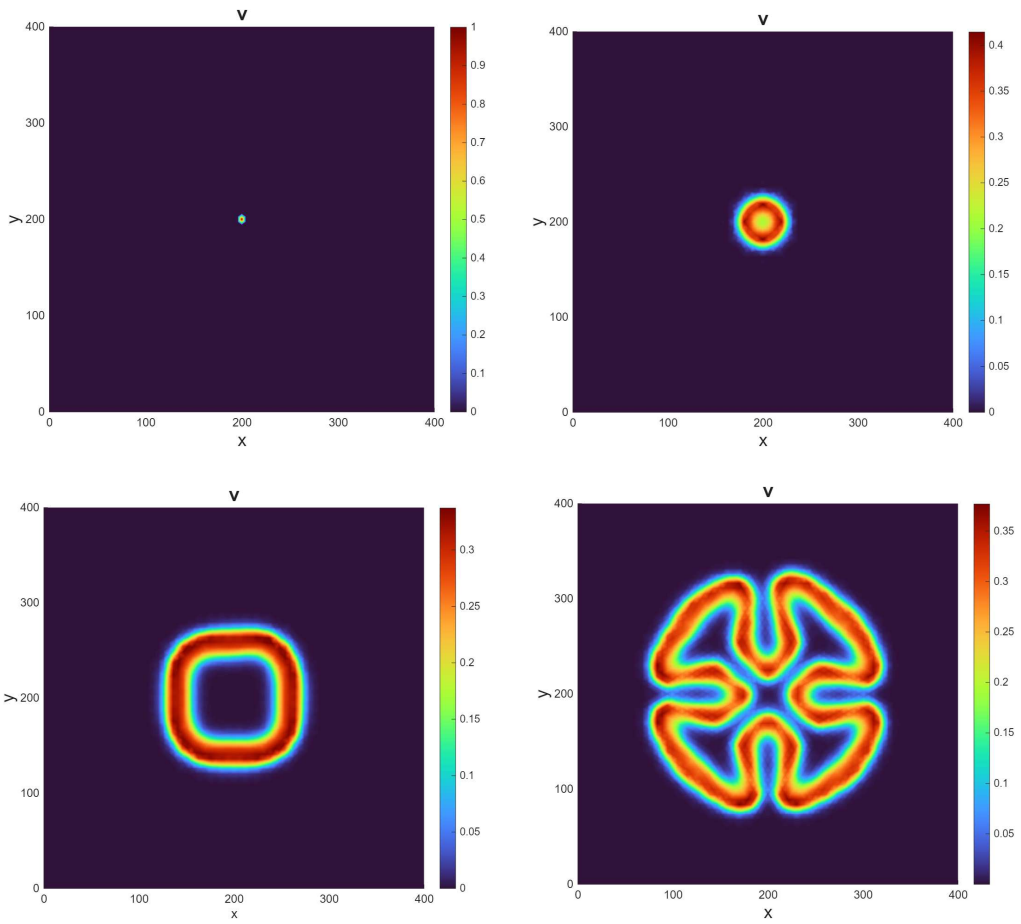

Figure 4.7: Solution of the 2D Gray-Scott system: Values of v at times $t = 0$, $t = 100$, $t = 1000$ and $t = 2000$ with FEM

the time evolution and pattern formation dynamics of the system. The consistency across all three classical solvers provides confidence in the correctness of the reference solution.

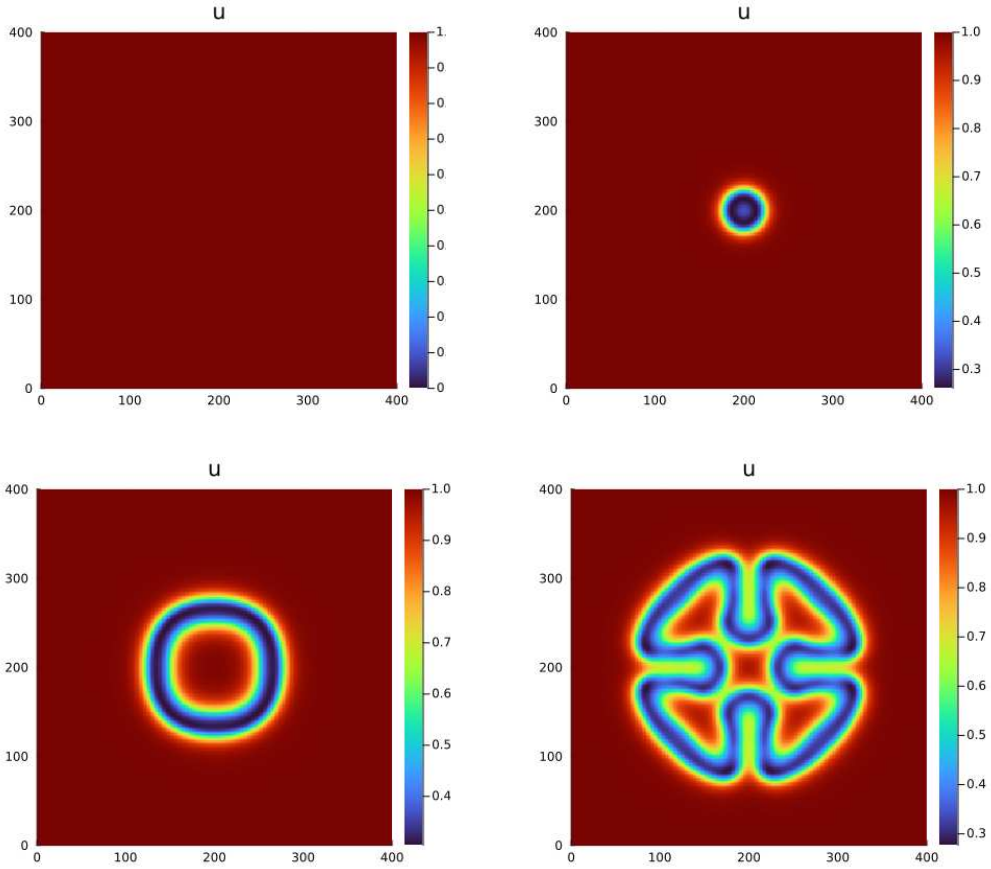

Figure 4.8: Solution of the 2D Gray-Scott system: Values of u at times $t = 0$, $t = 100$, $t = 1000$ and $t = 2000$ FVM

It is important to remark that the spatial patterns arising from reaction-diffusion systems of this type are highly sensitive to variations in model parameters, boundary conditions, and spatial discretisation. In particular, the solution exhibits mesh sensitivity, which highlights the need for careful selection of numerical parameters in problems of this kind.

Finally, the approximated solution of the PINN solver is provided. A first solution is computed using Vanilla PINNs, and a second solution is provided using an implementation of Hard PINNs, to strictly impose initial conditions. Both the networks are implemented in MATLAB using the Deep Learning Toolbox.

The architecture of the Vanilla PINN consists of 8 fully connected layers of 60 neurons, with hyperbolic tangent activation functions. The net is trained using an Adam optimizer for the first 100 epochs, with a learning rate of 1e-4 and then L-BFGS optimizer for 1400 epochs to further reduce the loss. The network computes the solution on 10000 collocation points, 2000 initial condition points and 2000 boundary condition points. All the collocation points are sampled randomly.

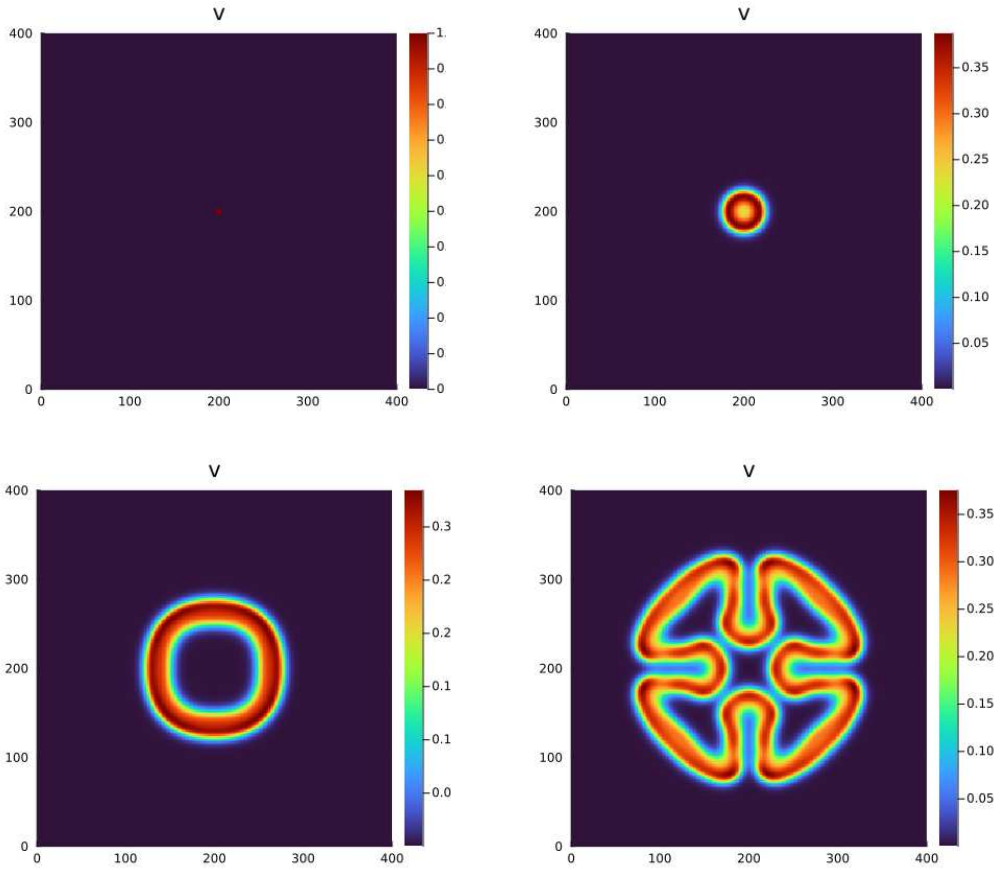

Figure 4.9: Solution of the 2D Gray-Scott system: Values of v at times $t = 0$, $t = 100$, $t = 1000$ and $t = 2000$ with FVM

The simulation ran for 5 hours and 20 minutes on a medium-power laptop. The results show a convergence of the loss function, proving that minimization is effective. Looking at the values of the results, though, it can be seen that no pattern arises. This outcome suggests that, despite successfully minimising the loss, the network converged to a local minimum of the highly non-convex loss landscape, rather than the physically meaningful global minimum that would yield the correct spatiotemporal evolution of the system. This is a proof of the struggles that PINNs can encounter when dealing with stiff problems, with big gradients and highly non-convex loss functions. A further critical issue is observed in the early time steps: the initial condition is not correctly enforced by the Vanilla PINN. Since systems of this type are highly sensitive to initial conditions, because small deviations in the prescribed values can lead to fundamentally different long-term dynamics, the failure to satisfy them accurately results in a physically inconsistent approximation from the very first instants of the simulation. This highlights a well-known limitation of the soft-constraint approach adopted by Vanilla PINNs, in which initial and boundary conditions enter the loss function as penalisation terms weighted by user-defined coefficients, rather than being imposed exactly. The inability to rigorously enforce these conditions represents a fundamental shortcoming when dealing with stiff, pattern-forming systems.

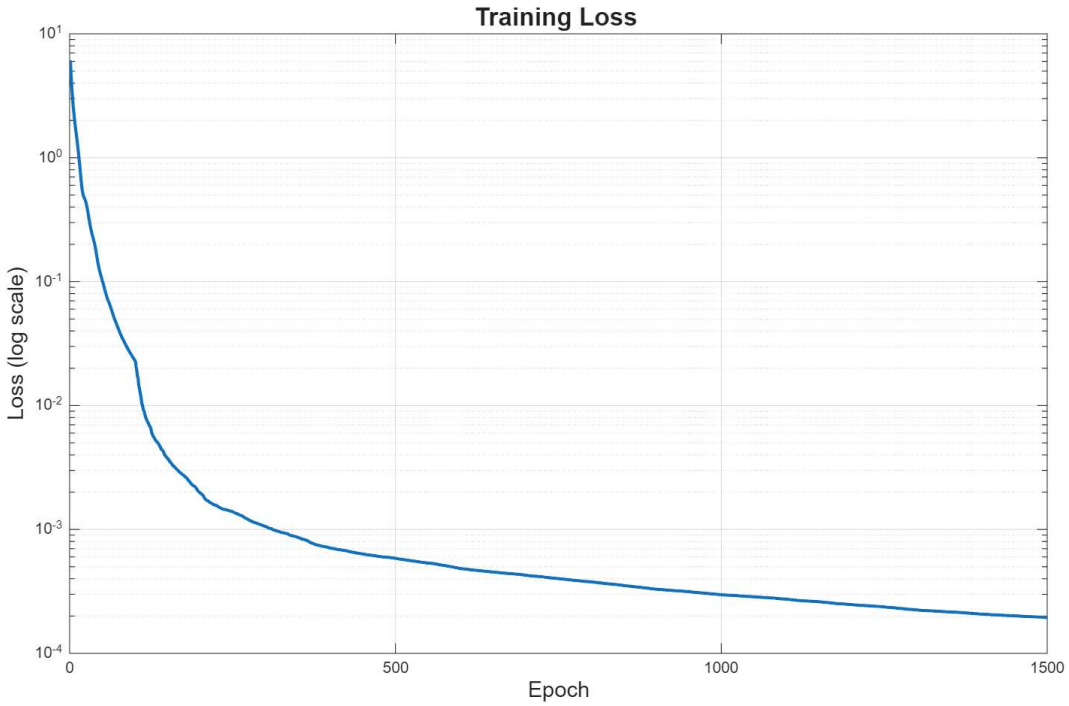

Figure 4.10: Training loss for Vanilla PINN

Motivated by the failure of the Vanilla PINN to enforce the initial condition, a Hard PINN formulation was subsequently employed, in which initial and boundary conditions are imposed exactly.

Figure 4.13 highlights the initial condition computed by the PINN. Figure 4.14 shows the error between the predicted and exact initial condition.The error displayed is quite small, meaning that the network effectively enforced the required condition,

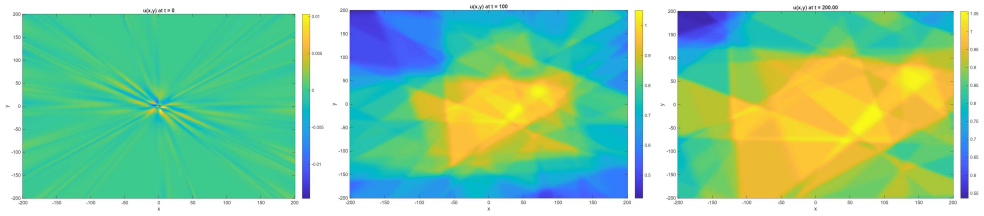

Figure 4.11: Solution of the 2D Gray-Scott system: Values of u at times $t = 0$, $t = 100$, and $t = 200$ obtained with Vanilla PINNs

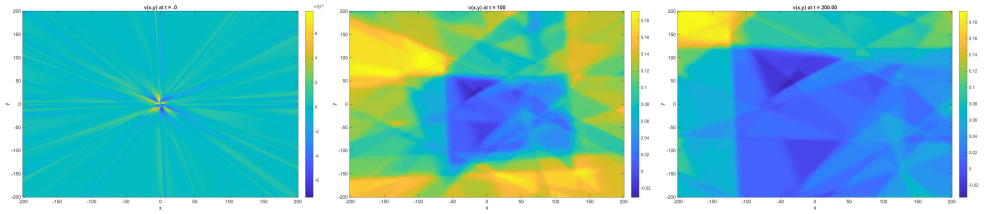

Figure 4.12: Solution of the 2D Gray-Scott system: Values of u at times $t = 0$, $t = 100$, and $t = 200$ obtained with Vanilla PINNs

in contrast to the Vanilla formulation.

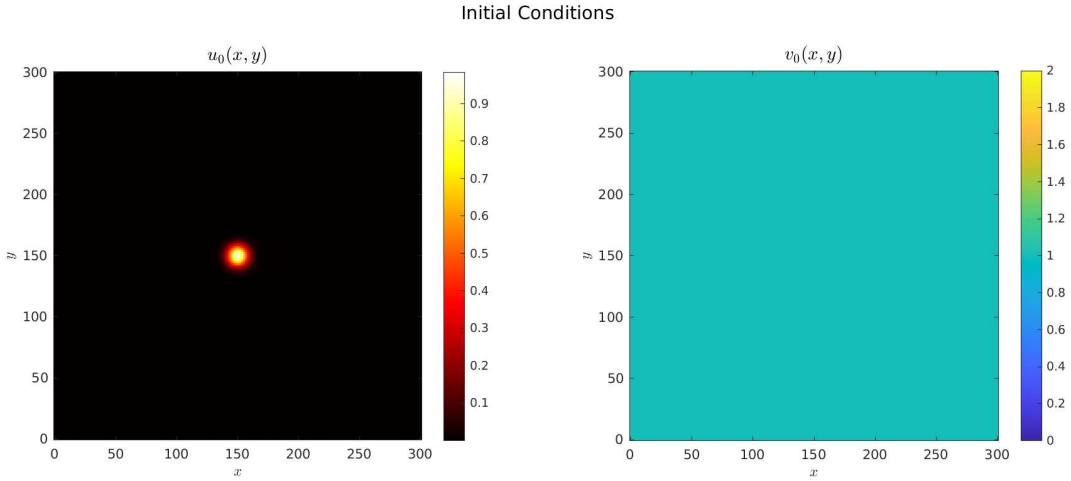

Figure 4.13: Initial conditions for 2D Gray-Scott equation obtained with hard PINN

However, despite the correct initialisation, the solution fails to evolve as expected and no pattern formation is observed at later times. Figure 4.15 shows the evolution of the values of u and v respectively from time 0 to 100.

Regarding the loss function, figure 4.16 shows that the value of the loss is very low throughout the whole simulation, of the order of 10^{-3}, fluctuating in a range of one order of magnitude.

Results for this second case study show that PINNs are not currently able to correctly solve the Gray-Scott system, which presents many challenges due to the sharp gradients, and highly non-convex loss landscape. In particular, it has shown that Vanilla PINNs fail to enforce initial conditions accurately, leading to physically

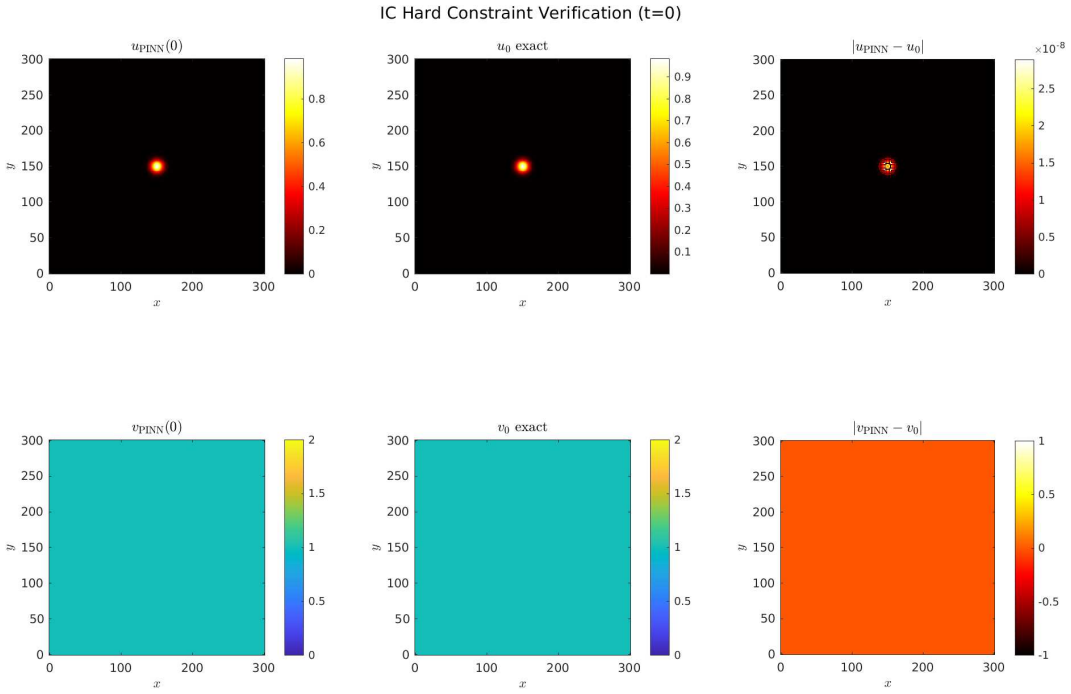

Figure 4.14: Error between the imposed initial conditions and the one predicted by the Hard PINN

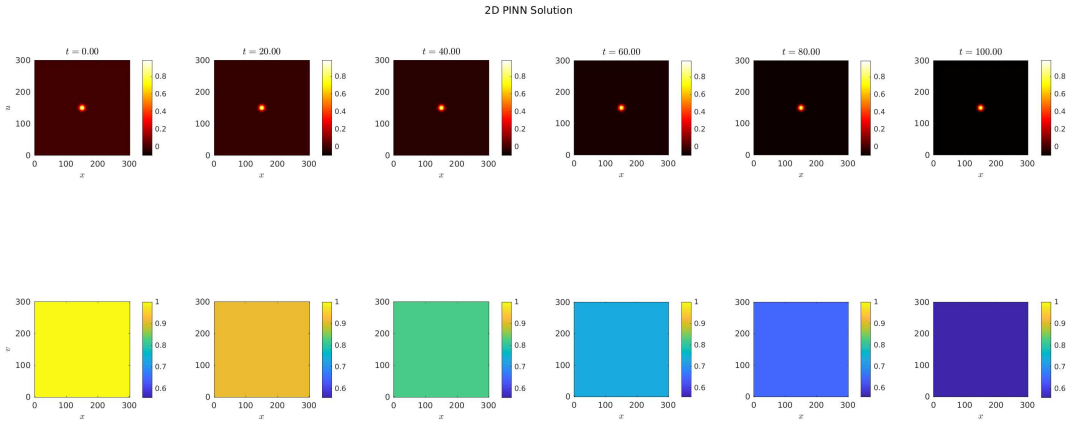

Figure 4.15: Snapshots of the solution of Gray-Scott using Hard PINN

Figure 4.16: Values of the loss of Hard PINN

incorrect solutions from the first time steps. While Hard PINNs are able to overcome this problem, they still fail in replicating the long-term dynamics of the system, suggesting that the difficulty does not only reside in replicating the prescribed conditions, but is a broader inability to capture the spatio-temporal structure of the solution. These findings are consistent with the growing body of literature highlighting the limitations of PINNs when applied to stiff, multi-scale and pattern-forming systems, and motivate the development of more advanced training strategies and network architectures.

Chapter 5

Conclusions

In this work we have introduced PINNs, explaining the mechanism and the main challenges that their employment presents to this day. We tested its efficiency and capability by solving two problems of increasing biological and mathematical complexity. The results show a good performance of PINNs on smaller domains and smoother solutions, but highlight their struggle in learning solutions to stiff problems, especially when trained in long time simulations.

At the current state of the art, PINNs prove to be a powerful tool for data-driven problems, such as the inference of equation parameters from known data. They also represent a promising approach for producing meshless solvers for PDEs, having produces positive results in some simpler benchmark problems; however, several aspects still need to be addressed before they can be considered reliable. This work aims to investigate the current possibilities and limitations of PINNs when employed as meshless solvers.

A first aspect to consider is that of the lack of a general convergence theory. Theoretical results on PINNs error bounds and convergence are available, but only for specific cases and applications. Therefore, it is not possible to know a priori whether the network will converge. Furthermore, the network could converge to a local minimum of the loss, with the result that the loss is minimized but the error between the approximate solution and the exact solution could remain large. For this reason, any solution produced by a PINN has to be validated through comparison with a known reference.

Some simulations are run to show the efficacy of PINNs in standard benchmark problems, and to provide proofs of the relevance of strict enforcement of initial condition. A first case study highlighted the potential of this methodology. The results confirm that PINNs are a viable meshless alternative for well-posed, low-dimensional problems. A second case study, focusing on a relevant problem in biology, revealed the current limitations of PINNs. In stiff Partial Differential Equations, where solutions exhibit sharp gradients, PINNs struggle to balance the three different loss components, leading to an unbalanced minimization of the total loss function. Furthermore, this problem involves a wide range of scales in the solution. Theory

shows that PINNs suffer from spectral bias, meaning that they tend to learn low-frequency solutions before high-frequency ones. In the Gray-Scott case, the PINN failed to capture the higher-frequency components that play a relevant role in pattern formation. A possible remedy to this bias is the implementation of Fourier Feature Embeddings, a method that has proven to be effective in certain cases, though its parameters must be chosen carefully through a trial and error procedure, in order to produce the desired results.

Another important aspect is that of the enforcement of initial conditions. In contrast to traditional numerical methods, where initial conditions are always imposed exactly, Vanilla PINNs only seek an approximation of them. Our simulation using a Vanilla PINN has showed that this soft minimization approach may not be sufficient to guarantee that initial and boundary conditions are correctly reproduced. In problems where the initial state strongly influences the subsequent temporal evolution of the system, such as in the Gray-Scott equation but also in many other biological applications, this failure of the PINN hinders the possibility of finding a good approximation for the solution over the rest of the space-time domain. The failure to enforce initial conditions also reveals another fundamental limitation: the lack of causality. Since the network is trained simultaneously on collocation points across the entire spatio-temporal domain, loss minimization occurs across all time frames at once and does not respect the causal structure of the problem. This could strongly impact the outcome of the solution for time-dependent systems. Since constraints cannot be directly embedded into a neural network, the only possible approach to overcome this problem is to incorporate them into the architecture itself. In the case study, a hard constrained formulation of PINNs revealed to be able to satisfy the initial condition with very low error. However, this formulation of the Hard PINN was unable to capture the subsequent emergence and evolution of patterns, indicating that the difficulty is not merely one of constraint satisfaction but reflects a deeper limitation in how PINNs handle the multi-scale, stiff dynamics of reaction-diffusion systems.

A further difficulty concerns constraints enforcement. Current hard-constrained formulations, which incorporate conditions directly into the network structure, remain highly restrictive and difficult to generalize. Ongoing research focuses on identifying optimal transformations for the functions that govern the embedding of the conditions into the network structure to enhance performance and generalisability. Future research on PINNs should focus on the development of more flexible constrained architectures. Lastly, it is important to note that the training of the network requires very long simulation times. The computational cost of each iteration of the PINN is very high and scales with the network size. It is worth considering, though, that once the network is trained, the evaluation of the solution at any point in space-time is extremely fast. The possibility of accessing an already trained network to evaluate the solution at any point may represent a promising direction for future exploitation of this methodology.

Bibliography

[1] Alan Turing. "The Chemical Basis of Morphogenesis (1952)". In: *The Essential Turing*. Ed. by B J Copeland. Oxford University Press, Sept. 2004. ISBN: 9780198250791. DOI: 10.1093/oso/9780198250791.003.0022. eprint: https://academic.oup.com/book/0/chapter/355747318/chapter-pdf/43817146/isbn-9780198250791-book-part-22.pdf. URL: https://doi.org/10.1093/oso/9780198250791.003.0022 (cit. on p. 3).

[2] B.J. Walker, A.K. Townsend, and A.K. et al. Chudasama. "VisualPDE: Rapid Interactive Simulations of Partial Differential". In: *Bulletin of Mathematical Biology* 85.11 (2023) (cit. on p. 9).

[3] W.S McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bulletin of Mathematical Biology* 52.1/2 (1990), pp. 99–115 (cit. on p. 11).

[4] D. P. Kingma and J.L. Ba. "ADAM: a method for stochastic optimization". In: *ICLR*. 2015 (cit. on p. 14).

[5] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations". arXiv:1711.10561. 2017 (cit. on p. 15).

[6] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. "Efficient training of physics-informed neural networks via importance sampling". In: *CoRR* abs/2104.12325 (2021) (cit. on p. 18).

[7] L. Lu, X. Meng, Z. Mao, and G. Karniadakis. "DeepXDE: A Deep Learning Library for Solving Differential Equations". In: *SIAM Review* 63 (1 2021), pp. 208–228 (cit. on p. 18).

[8] Nathan Doumèche, Gérard Biau, and Claire Boyer. *On the convergence of PINNs*. 2026. arXiv: 2305.01240 [math.ST]. URL: https://arxiv.org/abs/2305.01240 (cit. on p. 19).

[9] Yeonjong Shin, Jérôme Darbon, and George Em Karniadakis. "On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs". In: *Communications in Computational Physics* 28.5 (2020), pp. 2042–2074. ISSN: 1991-7120. URL: http://dx.doi.org/10.4208/cicp.OA-2020-0193 (cit. on p. 19).

[10] Marius Zeinhofer, Rami Masri, and Kent-André Mardal. *A Unified Framework for the Error Analysis of Physics-Informed Neural Networks*. 2024. arXiv: 2311.00529 [math.NA]. URL: https://arxiv.org/abs/2311.00529 (cit. on pp. 19, 24).

[11] 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. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2021.110768. URL: https://www.sciencedirect.com/science/article/pii/S002199912100663X (cit. on p. 20).

[12] A. D. Jagtap and Karniadakis G.M. "Extended Physics-Informed Neural Networks(XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations". In: *Computer Physics Communications* 28.5 (2020), pp. 2002–2041 (cit. on p. 21).

[13] Mingsheng Peng, Hesheng Tang, and Yingwei Kou. "Adversarial and self-adaptive domain decomposition physics-informed neural networks for high-order differential equations with discontinuities". In: *Engineering Applications of Artificial Intelligence* 145 (2025), p. 110156. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2025.110156. URL: https://www.sciencedirect.com/science/article/pii/S0952197625001563 (cit. on p. 21).

[14] Levi D. McClenny and Ulisses M. Braga-Neto. "Self-adaptive physics-informed neural networks using a soft attention mechanism". In: *Journal of Computational Physics* 474 (Feb. 2023), p. 111722. DOI: 10.1016/j.jcp.2022.111722. URL: http://dx.doi.org/10.1016/j.jcp.2022.111722 (cit. on p. 22).

[15] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. "Respecting causality for training physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 421 (2024), p. 116813. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2024.116813. URL: https://www.sciencedirect.com/science/article/pii/S0045782524000690 (cit. on p. 23).

[16] Yeonjong Shin, Zhongqiang Zhang, and George Em Karniadakis. *Error estimates of residual minimization using neural networks for linear PDEs*. 2023. arXiv: 2010.08019 [math.NA]. URL: https://arxiv.org/abs/2010.08019 (cit. on p. 24).

[17] Johannes Müller and Marius Zeinhofer. "Notes on Exact Boundary Values in Residual Minimisation". In: *Proceedings of Mathematical and Scientific Machine Learning*. Ed. by Bin Dong, Qianxiao Li, Lei Wang, and Zhi-Qin John Xu. Vol. 190. Proceedings of Machine Learning Research. PMLR, 15–17 Aug 2022, pp. 231–240. URL: https://proceedings.mlr.press/v190/muller22b.html (cit. on p. 24).

[18] Baoli Hao, Ulisses Braga-Neto, Chun Liu, Lifan Wang, and Ming Zhong. *Stability in Training PINNs for Stiff PDEs: Why Initial Conditions Matter*. 2025.

arXiv: 2404.16189 [math.NA]. URL: https://arxiv.org/abs/2404.16189 (cit. on p. 25).

[19] S. Berrone, C. Canuto, M. Pintore, and N. Sukumar. "Enforcing Dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks". In: *Heliyon* 9.8 (2023), e18820. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2023.e18820. URL: https://www.sciencedirect.com/science/article/pii/S2405844023060280 (cit. on p. 25).

[20] Lu Lu, Raphaël Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G. Johnson. "Physics-Informed Neural Networks with Hard Constraints for Inverse Design". In: *SIAM Journal on Scientific Computing* 43.6 (2021), B1105–B1132. DOI: 10.1137/21M1397908. eprint: https://doi.org/10.1137/21M1397908. URL: https://doi.org/10.1137/21M1397908 (cit. on p. 25).

[21] Xi'an Li, Jiaxin Deng, Jinran Wu, Shaotong Zhang, Weide Li, and You-Gan Wang. "Physical informed neural networks with soft and hard boundary constraints for solving advection-diffusion equations using Fourier expansions". In: *Computers Mathematics with Applications* 159 (2024), pp. 60–75. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2024.01.021. URL: https://www.sciencedirect.com/science/article/pii/S0898122124000348 (cit. on p. 26).

[22] N. Sukumar and Ankit Srivastava. "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114333. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.114333. URL: https://www.sciencedirect.com/science/article/pii/S0045782521006186 (cit. on p. 26).

[23] Florian Cajori. "The Early History of Partial Differential Equations and of Partial Differentiation and Integration". In: *The American Mathematical Monthly* 35.9 (1928), pp. 459–467. DOI: 10.1080/00029890.1928.11986877. eprint: https://doi.org/10.1080/00029890.1928.11986877. URL: https://doi.org/10.1080/00029890.1928.11986877.

[24] Haïm Brezis and Felix Browder. "Partial Differential Equations in the 20th Century". In: *Advances in Mathematics* 135.1 (1998), pp. 76–144. ISSN: 0001-8708. DOI: https://doi.org/10.1006/aima.1997.1713. URL: https://www.sciencedirect.com/science/article/pii/S0001870897917138.

[25] Philip K. Maini and Thomas E. Woolley. "The Turing Model for Biological Pattern Formation". In: *The Dynamics of Biological Systems*. Ed. by Arianna Bianchi, Thomas Hillen, Mark A. Lewis, and Yingfei Yi. Cham: Springer International Publishing, 2019, pp. 189–204.

[26] B. Macukow. "Neural Networks - State of Art, Brief History, Basic Models and Architecture". In: *IFIP International Federation for Information Processing*. 2016.

[27] S. Sharma, S. Sharma, and A. Athaiya. "Activation functions in neural networks". In: *International Journal of Engineering Applied Sciences and Technology* 4.12 (2020), pp. 310–316.

[28] D. C. Liu and J. Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45 (1989), pp. 503–528.

[29] N. Yadav, A. Yadav, and M. Kumar. *An Introduction to Neural Network Methods for Differential Equations*. Springer, 2015.

[30] I. Hariri, A. Radid, and K. Rhofir. "Physics-Informed Neural Networks Methodology for the Resolution of Some Turing's Type Models". In: *Advanced Mathematical Models & Applications* 10.1 (2025), pp. 61–80.

[31] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. M. Karniadakis. "Physics-informed neural networks (PINNs) for fluid mechanics: a review". In: *Acta Mechanica Sinica* 37.12 (2021), pp. 1727–1738.

[32] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J.M. Siskind. "Automatic Differentiation in Machine Learning: a Survey". In: *Journal of Machine Learning Research* 18 (2018), pp. 1–43.

[33] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 403 (2023), p. 115671.

[34] Yeonjong Shin, Jérôme Darbon, and George Em Karniadakis. "On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs". In: *Communications in Computational Physics* 28.5 (2020), pp. 2042–2074. ISSN: 1991-7120. DOI: 10.4208/cicp.oa-2020-0193. URL: http://dx.doi.org/10.4208/cicp.OA-2020-0193.