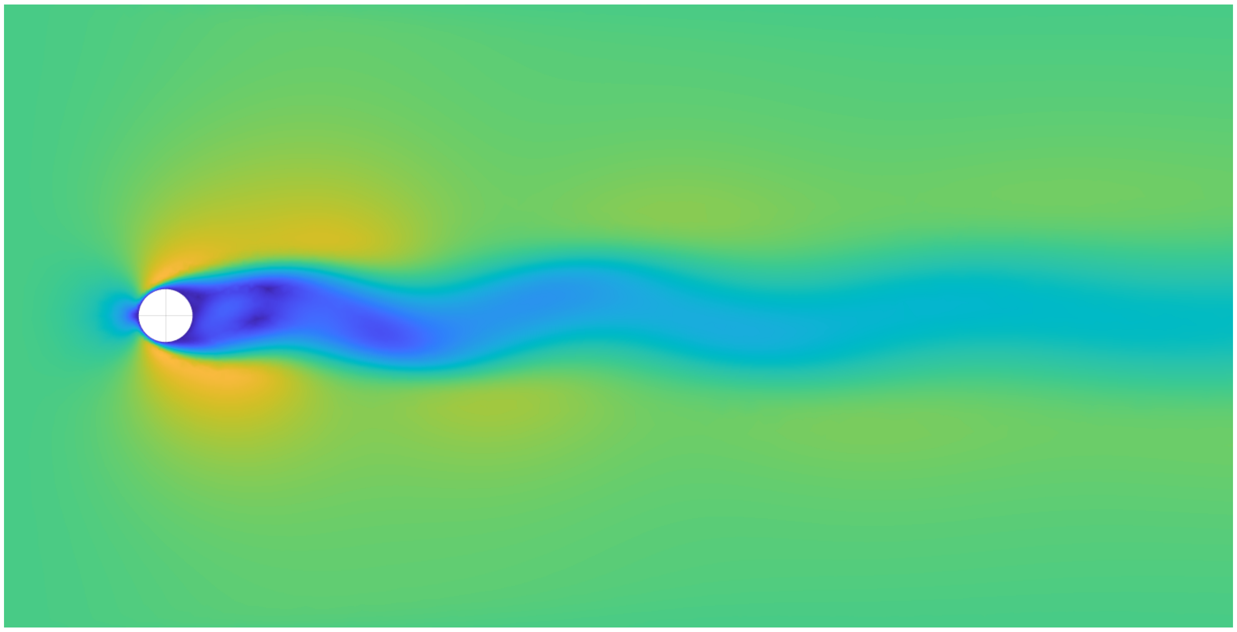
Physics Informed Neural Networks, A new approach to the solution of Partial Differential Equations

Master project



Fernando Wangüemert Guerra

Supervised by
Matthias Moller

June 19, 2021

To my parents. Always there.
Thank you

Contents

1 Introduction

With the considerable growth of available data and computing resources, recent advances in machine learning and data analytics have yielded transformative results across a wide range of scientific disciplines, including areas like image recognition, natural language processing or genomics [10],[16]. However, in the course of analyzing complex physical behaviour systems, the cost of data acquisition is prohibitive. The prior data to perform any analysis needs to be obtained via numerical methods or through experimental techniques [5]. The second option is to work in a small data regime, this is, to work with partial information that would make the method lack robustness and fail to provide any guarantees of convergence.

At first sight, this task of training a deep neural network (NN) to accurately identify a nonlinear map seems at best naive. The main advantage of a Physics Informed Neural Network (PINN) is that the physical laws that would constrain the space of solutions are the tool that guides the optimization process towards the desired solution.

The study of physical systems with this approach offers a wide range of possibilities concerning the different tools that the field of machine learning offers. These tools that were originally designed to solve a different type of problems have been updated and nowadays they have become very versatile. In our case, we will be using automatic differentiation [1] to describe the space and time evolution of a certain problem, this is, the physical laws that govern the simulation.

We will see that this method, first developed by Maziar Raissi, Paris Perdikaris, and George Em Karniadakis in [17], [15] and [18] (under adequate conditions) offers results with an error magnitude comparable to advanced discretization schemes that have been developed and consequently optimized during the last 50 years.

The main objective then is to take advantage of the adaptability of NN as a universal function approximator, [4] to perfom these physical simulations in different scenarios bypassing the data scarcity. This flexibility makes the method suitable for simulating incompressible flows ranging from laminar to turbulent regimes. We refer to these specific PINNs for Navier-Stokes flows as NSFNets [6], [18].
The objective of this master project is to develop a Physics Informed Neural Network model [15] for the efficient design optimization of wind turbines and to integrate it into the Monolith AI platform. This will include the creation of two separate PINN models for incompressible fluid flow and (linear) elasticity and their coupling to a fluid-structure interaction (FSI) model.

The Monolith AI framework provides variational auto-encoder (VAE) capabilities which allow to represent a family of similar shapes, e.g., three-bladed wind turbines, with a small number of relevant design parameters that enable the efficient creation of design variations. The main novelty of this project is to combine the PINN-based FSI models with the VAE geometry modelling engine to enable efficient and intuitive design optimization of wind turbines.

The structure of this report as follows; Firstly, we show how the numerical simulation has been constructed, giving a certain insight in the problem set-up and the conditions under which we will work. This simulating paradigm is then applied to explore the behaviour of different example PDEs under a certain range of boundary conditions. We then perform a convergence study on the PDEs that were studied before. Finally, the advantages and challenges of the method will be presented summarizing our findings.

2 Motivation

Today's engineering design workflows are typically based on established computer-aided technologies (CAx) that support engineers in modelling geometric designs (CAD) and investigating their properties (CAE) with the aid of, e.g., computational fluid dynamics (CFD), computational structure mechanics (CSM) or acoustics simulations.

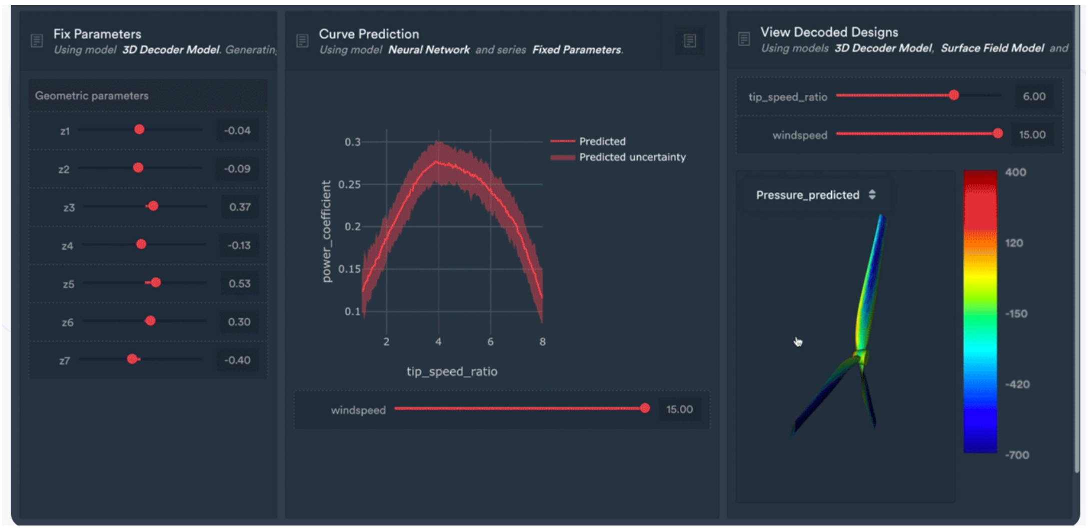
Designs can then be optimized either manually by the designer or computer-driven to, say, minimize the weight, the material, production and maintenance costs and the ecological footprint while at the same time maximizing the performance and durability.

Designing complex engineering products such as wind-turbines is a nontrivial task that often takes weeks to months with today's tools based on classical numerical methods for solving PDE problems. The classical workflow set to develop a new product starts with a design development process on which a number of simulations is performed. In a next stage prototypes are built and physical tests are run in them. Only after all these steps are finished, the performance of the product can be validated. This lengthy iterative process is very expensive because in each design iteration, no re-use of previous calculations is possible. All the computational effort has to be made without any use of the prior data available. This fact makes multi-objective optimization very difficult to achieve. As soon as the number of features of a product that affect it's overall performance is too numerous, the optimization process becomes practically unfeasible.

The motivation of the project then is, to exploit the machine learning capabilities to carry out an efficient optimization process that covers: First, simplifying the design optimization process so that all the design features of the product only depend on a certain small number of parameters. This technique can be performed through what is called a Variational Auto-Encoder (VAE) [9] that reduces all design aspects of a product to a desired number of latent parameters (left part of figure 1).

If, added to this design simplification, we can perform a PINN simulation that covers the whole range of possible values of the latent parameters,the resulting model would only need one expensive off-line training that would cover the entire design process going through all the design possibilities, making it considerably more efficient. This way of working, takes advantage of the capacity that a neural network has to adapt the prediction of a simulation to a similar scenario i.e, similar latent parameters, so that the design iteration process is all done at once.

Once the network optimization is finished, the different simulation results to every design option are available to the user without any processing time. This fact deeply saves time, costs and computational effort with respect to the classical workflow process.



Geometric parameters

z1	-0.04
z2	-0.09
z3	0.37
z4	-0.13
z5	0.53
z6	0.30
z7	-0.40

windspeed 15.00

tip_speed_ratio 6.00

windspeed 15.00

Figure 1: Monolith AI design optimization example

3 Neural Networks and Machine Learning:

Artificial neural networks (ANN) are a computational model inspired in the behaviour observed in biological neurons. Each connection, as in the synapses that happen in a biological brain, can transmit a signal to other neurons. In the case of an artificial neuron, it receives an input, processes it via a non-linear operator and then produces an output that will serve as an input to other neuron.

The "signal" at a connection is a real number that has been computed as a result of applying a nonlinear or linear operator to the scalar product of the inputs by a set of weights. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. The simplest example of an artificial neuron was proposed by Frank Rosenblatt in 1958, called the Perceptron [21]. This model gave rise to the field of the artificial neural networks known nowadays. The perceptron has the behaviour shown in figure 2:

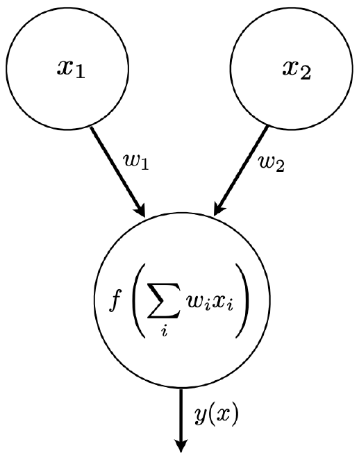


Figure 2: Perceptron example (from [21])

The input nodes x_i receive a certain package of information (in this case a real number), then the inputs are multiplied with a weighting factor w_i and summed up. Finally the integrated input is mapped through some non-linear or linear function (It is important to remark that the identity function is also a possibility),

$$f\left(\sum_i w_i x_i\right)$$

The characteristics of this nonlinear operator will undoubtedly influence the main features of the network. For instance, if this nonlinear operator equals:

$$f(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{x} \text{ is preferred stimulus} \\ -1 & \text{if } \mathbf{x} \text{ is any other stimulus} \end{cases}$$

This perceptron would be performing a classification task with respect to the input features x_i. The main question that arises is how to select the set of weights optimal to the task at hand. This weights constitute the set of optimizable parameters of the network. This means that with an error function depending on them, we can choose a set of weights such that the error function is minimized. It will be seen in section 7 that in our case, the error function constitutes the set of physical laws and boundary conditions that the network must follow.

Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. This notion of multiple layers and different perceptrons per layer is also called Multi Layer Perceptron (MLP).

As stated before, it is possible that the output of a neuron may be the input of another one, constituting a superposition of layers before delivering an output. This set of layers that are neither the first nor the last, are called hidden layers. These layers deeply improve the complexity and adaptability of the MLP and allow it to mimic the behaviour of a wide range of functions. Making it therefore ideal to the pursue the objectives initially set in [17].

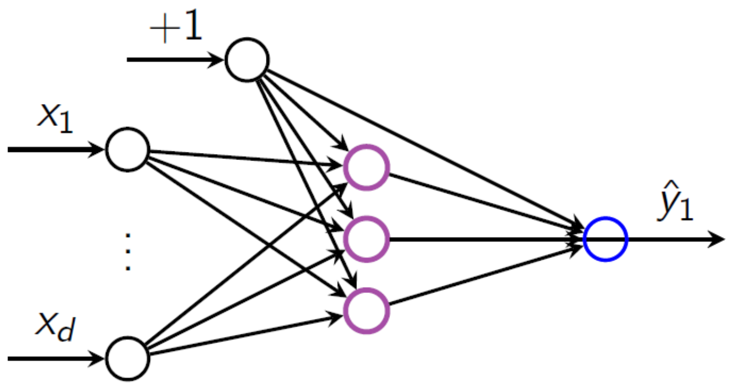


Figure 3: MLP with one hidden layer (purple)

3.1 Training

The process of training a neural network mainly consists in adjusting the values of the weights and biases that the network depends on, so that it fulfills it's objectives the best possible way. For this purpose, we define a loss function that represents the criteria on which the network is going to change it's parameters. It maps a certain weight selection to a cost value. The objective then to get the best possible regression or classification, is to minimize the cost associated with it that has been established by the loss function.

The widely used way to optimize this loss function consists in an optimization with gradient based methods. These methods consist in learning by gradient descent in the space of parameters. Starting with an initial set of weights, the optimized new weight after an iteration of the gradient descent method reads:

$$w_{ij}^{n+1} \leftarrow w_{ij}^{n} - \eta \frac{\partial E}{\partial w_{ij}}\bigg|_{\omega_{ij}(n)}$$

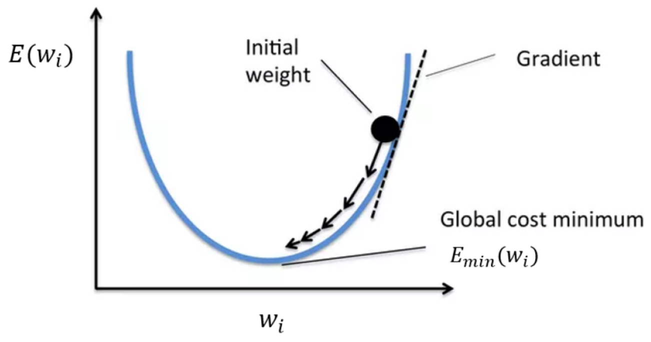


Figure 4: Illustrative figure of Gradient Descent [12]

where in this case, E is the error (cost) function and η is the learning rate. This learning rate is a tuning parameter in the optimization process that determines the step size at each iteration while moving the set of parameters towards the minimum of the loss function.

3.2 Universal Approximation theorem

One can show that a single hidden layer MLP is a universal function approximator [4], meaning that in can model any suitably smooth function, given enough hidden units, to any desired level of accuracy. This property of MLPs is key in the development of physics informed neural networks since the main objective is to aquire an analytical expression of the function that behaves as a certain PDE predicts, jointly with the boundary conditions.

The Universal approximation theorem implies that neural networks can represent any variety of different functions given the appropiate weights. This theorem is the cornerstone of the strategy that Maziar Raissi, Paris Perdikaris, and George Em Karniadakis take in [17] to construct PINNs.

4 Overfitting

When working with any ML problem, one of the main reasons the output solution fails to predict the desired results comes from using a model too complex for the difficulty of the problem itself.

While it is true that this issue concerning MLP optimization is one in many, it was decided to inlude it as a separate section because is is one of the problems that become automatically solved by the algorithm definition of a PINN. This will be further assesed in section 7.2.

Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. An example of this phenomena can be seen in figure 5

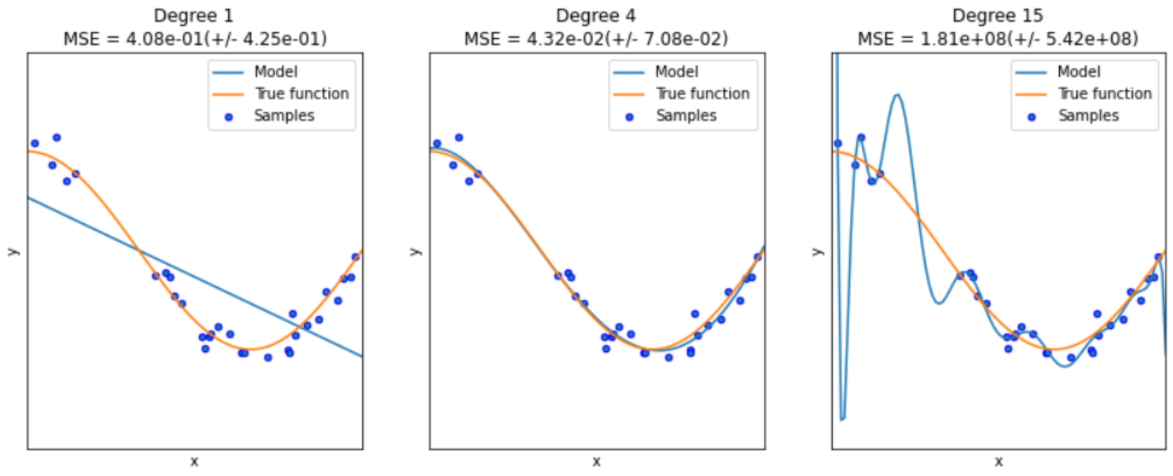


Figure 5: Overfitting illustration [12]

In this figure, a regression model has been constructed in order to mimic the behaviour of a function, in this case the orange coloured. If the model is too simple, the whole evolution of the function through space won't be detected. The test data and the training data both have big error in the model's prediction (c.f left fig of figure 5).

When the regression model has a certain level of complexity, a well-posed training algorithm with the training data will make the model capable to correctly predict the desired solution in the test data (c.f middle fig on figure 5).

The right sub-figure in figure 5 illustrates overfitting behaviour. The model correctly predicts all the points on which we have trained the model but fails to predict any other point. This is also known in machine learning as the bias-variance equilibrium. In an overfitted model, the bias of the model is very low but the variance (represented in the magnitud of the weights of the network) spikes.

As stated before, we will see in the following, that one of the advantages of Physics informed neural networks is that the construction of the algorithm hampers the possibilities that the given approximator is overfitted. The enforced physical law acts as a regularization agent throughout the whole domain.

There are more details on the insight of the machine learning algorithms during this master thesis but the main aspects of it have already been covered. With this short theoretical base, we will develop the problem set up of a PINN and posteriorly we will use the machine learning software TensorFlow to develop a physics Informed Neural network in which we will experiment with different problems and domains.

5 TensorFlow

All the programs developed during this thesis will use the TensorFlow platform [13] in order to achieve the desired results.

TensorFlow is a free and open-source software library for machine learning. It can be used across a wide range of disciplines within machine learning but has a particular focus on training and inference of deep neural networks.

Its main features are the possibility of working with a symbolic math library on the data and the automatic differentiation algorithms. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers and students push the state-of-the-art in ML and developers easily build and deploy ML powered applications. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 in 2015.

The single biggest benefit TensorFlow provides for machine learning development is abstraction. Instead of dealing with the low-level details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another or handling ways to construct your code more efficiently, the developer can focus on the overall logic of the application, the maths behind it. TensorFlow takes care of the details behind the scenes. It is true that for a master project the possibilities that TensorFlow offers may make the ANN look as a black box, on which only the inputs and outputs are showed in a more explicit way, but nevertheless this allows the student to focus on the applications that may be given to this tool. In our case, the use of TensorFlow makes the code substantially clearer and also makes the project feasible. Any project related to the machine learning field that didn't use a ML platform like TensorFlow would be slowed down months just to construct a set of functions that makes the user able to perform the necessary calculations. Furthermore, the TensorFlow platform is frequently optimized making it the best option to efficiently run your simulations.

The open-source platform allows one to build and train deep learning models easily using intuitive APIs like Keras. The use of these tools simplifies the resulting code extensively and consequently eases the debugging tasks.In this master project, the Tensorflow version that will be used is v.2.3.0 , accompanied by other packages as numpy with version 1.18.5 and TensorFlow probability version 0.11.0. This last package covers the implementation of the L-BFGS optimization algorithm that is introduced in section 7.1.2.

On the other hand, Tensorflow provides a low-level library too which provides more flexibility. Thus, all functionalities and services of the platform are fully customizable. This flexibility will allow us to change certain aspects of the Neural network that differ with respect to the classical examples of supervised learning. Mainly concerning the design of the cost function.

6 Problem Set up

The main difference between Physics Informed Neural Networks and other machine learning methods whose objective is to solve partial differential equations is, that in this approach, we leverage the capability of a neural network as an universal function approximator (section 3.2). In this setting we can assess the problem directly without making any prior assumptions (e.g linearization). We will exploit, as presented before, the advantages that automatic differentiation provides, to differentiate neural networks with respect to their input variables and model parameters (trainable variables) to finally obtain a physics informed neural network.

Such neural networks are logically constrained to respect any symmetry, invariance, or conservation principles originating from the physical laws that govern the observed data, as modeled by general time-dependent and nonlinear partial differential equations. It is important to note that this observed data may or may not be present in the presentation of the problem. The algorithm of the PINN is autonomous from any prior knowledge of the model's solution.

This simple yet powerful construction allows us to tackle a wide range of problems in computational science. From [17]:

> "It introduces a potentially disruptive technology leading to the development of new data-efficient and physics-informed learning machines and new classes of numerical solvers for partial differential equations".

During this report it will be remembered how little information about the physics of the problem is fed into the Neural Network itself, remarking the generalization capabilities of the method, applicable to any physical law or conservation property of the problem in question.

The general objective of the project then is to explore the possibilities of this new paradigm in computer simulations presented in [17].

During this master project we will focus on what has been called in literature: data-driven solution of partial differential equations. To this end, let us consider a parametrized and nonlinear partial differential equation of the general form:

$$u_t + \mathcal{N}[u; \lambda] = 0, \tag{1}$$

where $u(x,t)$ denotes the unknown solution and $\mathcal{N}[u;\lambda]$ is a nonlinear operator parametrized in this case by λ. This setup represents a wide range of scientific applications such as conservation laws, diffusion processes, kinetic equations or any other behaviour of any kind. Having this problem together with a set of initial and boundary conditions, the algorithm consists in short in: using the neural network as a universal function approximator, and set the boundary conditions (temporal and spatial) and the constraints of the nonlinear pde ($u_t + \mathcal{N}[u;\lambda] = 0,$) into the loss function. We will detail the specifics of this method in the next section.

7 Algorithm, PINN Approach

As said before, we are going to use the neural network as a universal function approximator that will have as input the spatial and temporal coordinates of the problem and deliver as an output a prediction on the value of the unknown in the point that the network was evaluated. Therefore, the MLP constructed constitutes an analytical expression that aproximates the solution of the PDE problem set. Once its weights have been optimized, it will represent an analytical approximation of the desired solution $u(x,t)$ of the problem. The main characteristics of the architecture of the neural network are shown in figure 6.

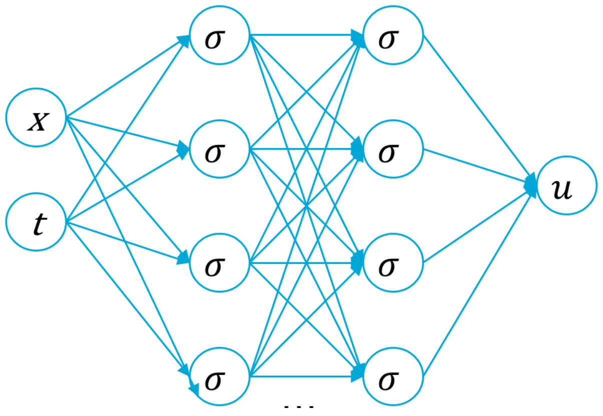


Figure 6: Standard characteristics of the MLP used in a PINN

The σ in this graph represent the activation function present in each layer.

What figure 6 represents is the typical structure that is applied to the MLP to approximate the analytical expression from $u(x,t)$. The MLP will have as many inputs of variables as variables has the solution (e.g for a 1D evolution of the heat equation the MLP would have 2 input variables, 1D space and time) and as many outputs as the problem's solution, in case of a scalar function the MLP will deliver one output which we will take to be the prediction of the value of $u(x,t)$ in the point in which it was evaluated.

The dimensions of the neural network in figure 6 show that the network has a certain number of hidden layers and that in each one, we can choose the number of "neurons" that each layer has. Each of the layers in the network is called "dense", when each of the neurons of the layer is connected to all of the other neurons of the previous and next layer respectively. The dimension of the network used in each problem is one of multiple parameters that can be changed in the PINN set up to solve the problem, with its consequences. The case showed in figure 6 is not representative of the MLP used for these applications, we observe a two layer network with 4 neurons per layer (2×4). In the first example shown in section 8.1 , the architecture used is (5×20).

The network architecture chosen has a deep effect in the error shown after optimization [17], in the convergence properties of the network and in the optimization time. Choosing it correctly results fundamental to achieve a correct result. In [17], an error study was performed for the burgers equation 1D for different network architectures. This study will be thoroughly explained in section 8.1 once all the PINN concepts have been set.

For the optimization process of this neural network,the technique proposed in [17] consists in enforcing the partial differential equation and boundary and initial conditions that define the problem through 2 different sets of points. It will only be in these finite sets of points where the conditions that define the problem will be enforced. In a first set of points called training points in[17] , the boundary and initial conditions will be enforced the following way; a part of the loss function is going to contain the expression [17] :

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 \tag{2}$$

This is, the mean square error of the difference between the value of the NN in a given set of boundary points, and the desired values set by the boundary condition in all those points.These training points are logically located at the boundary (spatial and temporal) of the considered domain. It is important to know that the training points on this supervised learning task do not require any prior knowledge on the solution of the problem.

This kind of square error is the classical structure of the loss function that has been constructed in learning models in the past. It has been observed during the initial stages of the code development in this master project that the convergence of the NN towards a function that follows the boundary conditions is very fast when these boundary conditions are dirichlet. This fact will therefore be studied in the future of the master project to see whether the optimization results faster when the network is trained first in the boundary and then in the rest of the domain.

7.1 Collocation points

In the second batch of points that are introduced in the algorithm, the physical law will be enforced. This means that we will enforce that the Neural Network must behave like the PDE sets in a finite set of points of the domain. This finite set of points is the one referred to in [17] as "Collocation Points".

We define $f(x, t)$ to be given by the equation 1; i.e.,

$$f := u_t + \mathcal{N}[u]$$

This way, the second and last part of the loss function is the mean square error of the value of the function $f(x, t)$ evaluated in each point of the collocation point set [17].Minimizing this error function to zero will mean that the NN behaves as the PDE sets in all the collocation points. Again, this network can be derived by applying the chain rule i.e using automatic differentiation tool that TensorFlow provides.

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

The global loss function then becomes the sum of these two [17], enforcing the NN to follow both the PDE and the boundary conditions :

$$MSE = MSE_u + MSE_f$$

We can find a scheme of this procedure in figure 7 constructed with the same structure as in [16]

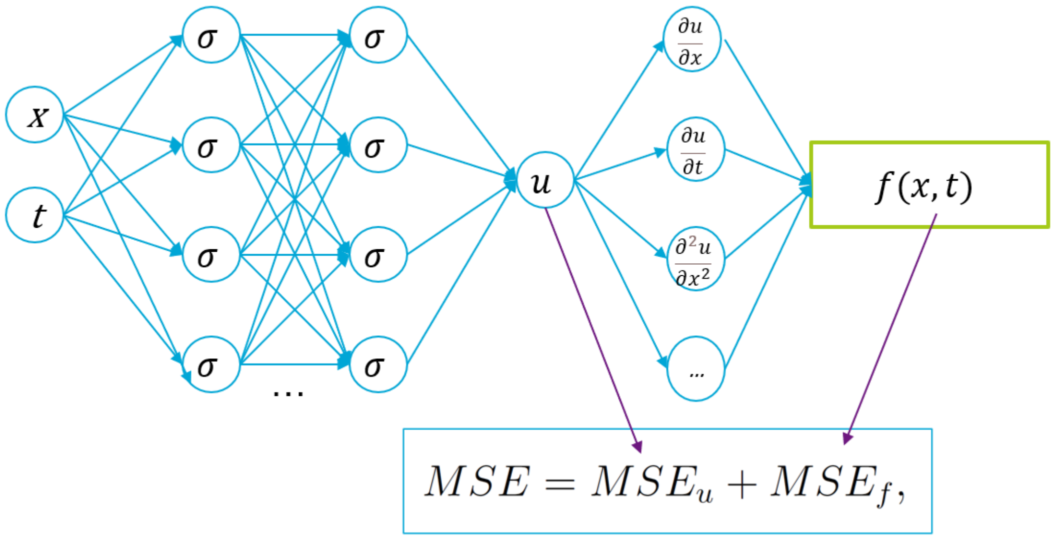

x
t
σ
...
u
$\frac{\partial u}{\partial x}$
$\frac{\partial u}{\partial t}$
$\frac{\partial^2 u}{\partial x^2}$
...
$f(x,t)$
$MSE = MSE_u + MSE_f,$

Figure 7: Global scheme of the PINN algorithm

Once the loss function has been defined, this loss value can be modified changing the trainable parameters (weights and biases) in the neural network. Therefore, this parameters will be optimized to reduce the loss value to a minimal possible level. In this training process it is important to highlight the robustness of the method with respect to the well known issue of over-fitting.

Specifically, the term that enforces the partial differential equation acts as a regularization mechanism that penalizes solutions that do not satisfy the equation. Therefore, a key property of physics informed neural networks is that they can be effectively trained using small data sets, a setting often encountered in the study of physical systems for which the cost of data acquisition may be prohibitively expensive.

For this training purpose, during the thesis there are mainly two optimizers that will be used in the study cases exposed: The first one, the Adam optimizer allows us to divide the set of collocations points in smaller groups (batches) in order to speed up the optimization process. On the other hand, the L-BFGS optimizer only allows in tensorflow to use it as a full-batch algorithm, this is, executing the optimization taking into account all the collocation points initially defined. In the next section, the main characteristics of both optimizers are defined.

7.1.1 Adam Optimizer

The Adam optimization algorithm [8] is an extension to stochastic gradient descent that has recently seen as a broader adoption for deep learning applications in computer vision and natural language processing.

Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training, whereas in the Adam Optimizer, the learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.The name Adam comes from Adaptive moment estimation. It will be mainly used when a full batch optimization is unfeasible due to the amount of input data.

7.1.2 L-BFGS Optimizer

The Limited-memory BFGS (L-BFGS) [22], is an optimization algorithm in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory. It is a popular algorithm for parameter estimation in machine learning.

The use that has been done in the literature [6] consists in starting the optimization with the Adam optimizer, and once the loss value is sufficiently small, proceed with the L-BFGS optimizer. This is because these quasi newton optimizers, as the newton optimizer, do have convergence problems in case the starting position isn't sufficiently near the minimum, but once convergence is achieved, they result significantly faster than the Adam algorithm.

This optimization algorithm isn't supported in the TensorFlow package but in the TensorFlow Probability package.Because of this, to use this optimizer with respect to to the weights of the neural network is necessary exporting the weight structure of the network into the algorithm in order for it to be useful. The implementation process of this optimizer has been done as in [14].

7.2 Peculiarities of the approach

A distinguishing fact about PINNS with respect to any other numerical simulation procedures is that in the former we optimize an analytical expression towards the hidden state $u(x,t)$. The output of the method is not a set of predictions in an aggregate of points. It is the analytical optimized solution that we use to predict the value of the property studied in any point.

In the algorithm, there is not a specific step in which, by definition a mesh creation or domain definition is necessary. The algorithm just needs a batch of points in which apply the conditions set in the loss function. This then means that the mesh generation effort isn't strictly necessary for the algorithm as it is for any other numerical discretization method. The meshes that will be seen during this report are used solely for data structuring and representation purposes.

All the universal function approximators used i.e neural networks, are defined in an unbounded domain $\mathbb{R}^N$ being N the number of inputs (coordinates) of the neural network. This fact makes the implementation of boundary conditions substantially easier if we compare it with other simulation techniques like the finite element method. The fact that the boundary conditions are enforced via the loss function makes their implementation very flexible (the implementation of periodic boundary conditions is trivial).

In the first paper published concerning PINNS [17], it was stated that while there was no theoretical guarantee that this kind of procedure converges to a global minimum i.e the solution of the partial differential equation together with it's boundary conditions, empirical evidence suggests that it the differential equation is well-posed the method is capable of achieving good prediction accuracy given a sufficiently expressive neural network architecture and a sufficient number of collocation points.

But the article published in October 2020, [20] provides proof that for linear second-order elliptic and parabolic PDEs, the sequence of minimizers strongly converges to the PDE solution in L_2.The theoretical proof of this fact can be found in all it's extent in [20] but it hasn't been developed in this literature review because if goes out of the scope of the project.

We will see that despite the fact convergence has been proven in a number of cases, in a practical case, convergence can result a troubling area in some scenarios. Nevertheless, a complete study on convergence in relation to the problem parameters (Network arquitecture / Collocation point structure ...) will be carried out during the second semester in the master project.

With the supporting theory explained, we proceed now to study some of the PDE examples from [17] , [6] and [3] in order to get more insight on the functionality of PINNs. It is important to remark this first report just constitutes a first view on the different examples that have been carried out to understand the behaviour of the algorithm. A more detailed study concerning the multiple parameters a PINN depends on (collocation points, network architecture) will be performed in these examples during the master project.

8 Study Cases

In this section we proceed to show the first preliminary simulations carried out in this master project. It is important to remark that a more thorough study will be performed in each of the study cases.

8.1 Burgers Equation

As an academic example, let us consider Burgers Equation. This equation arises in multiple areas of applied mathematics, including fluid dynamics concerning the formation of shock waves. It is a fundamental partial differential equation and can be derived from the Navier-Stokes equations for the velocity field by droping the pressure gradient term. For small values of the viscosity parametres, Burgers equation can lead to shock formation that is notoriously hard to resolve by classical numerical methods. We will study the following case:

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1,1], \quad t \in [0,1]$$

With the boundary conditions:

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

Let us define $f(x,t)$ to be given by:

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

and proceed to approximate $u(x,t)$ by a deep neural network. Below we give the python code for the construction of a simple burgers network to highlight the simplicity of the idea.

```
import tensorflow as tf

layers = [2, 20, 20, 20, 20, 20, 1]

model = tf.keras.Sequential()

model.add(tf.keras.layers.InputLayer(input_shape=(layers[0],)))

for width in layers[1:-1]:
     model.add(tf.keras.layers.Dense(width, activation12 =tf.nn.tanh,
     kernel_initializer="glorot_normal"))
model.add(tf.keras.layers.Dense(layers[-1],
activation = None,
kernel_initializer="glorot_normal"))
```

In this case the whole construction of the feed forward neural network has been done with the Keras tool provided by the TensorFlow platform. The structure of this Neural network with dense layers is stated in line 3 of the code as it was explained in section 7. The kernel initializer in the code has as a function initialize the values of the weights the network will start with. In this case, the initial weight set up corresponds to a set of samples drawn from a truncated normal distribution centered on 0. The selection of this initial weight structure is taken from [17] , [3] and [14].

Correspondingly, the physics informed neural network $f(x, t)$ takes the form:

```
def f_model(X, model):

    lambda_1 = tf.Variable([1.0])
    lambda_2 = tf.Variable([0.01/np.pi])

    u = model(X)
    u_x = tape.gradient(u, x)
    u_xx = tape.gradient(u_x, x)
    u_t = tape.gradient(u, t)

    f = u_t + lambda_1*u*u_x - lambda_2*u_xx

    return f
```

Then, as stated before, the shared parameters between the neural networks u and f can be learned by minimizing the mean squared error loss:

$$MSE = MSE_u + MSE_f$$

where the subscript u corresponds to the fact that that part of the loss function is evaluated in the boundary (training) points and the subscript f corresponds to the part of the loss function that is evaluated in the collocation points. In this case, the parameters used on the problem are the following; 100 training points, a network structure of 5×20 (5 hidden layers and 20 neurons per layer) and hyperbolic tangent activation functions. In the last hidden layer, the activation function will be the identity in order no to limit the range of values of the network to the interval $[0, 1]$. The optimization process will be performed with 3000 iterations on the Adam optimizer and another 3000 iterations via the L-BFGS optimizer. The collocation points are randomly selected within the points of the domain. In figure 8 we can observe the first of the two sets of points neccesary for the training of the Neural Network. This first set of points called training points [17] are responsible of enforcing the initial and boundary conditions in the network. The process goes as following: the network, (that is an analytical function) is evaluated in each of the points that appears in figure 8, then, each output is compared with the desired value at that point (the boundary and initial conditions that in this case, are both dirichlet).

The resulting output of these two operations is a set of 100 real numbers (100 training points) presumably different than 0 (The boundary and initial conditions are not initially satisfied). If then those 100 real numbers are individually squared and then the mean of the joint set is computed. The resulting number is a loss value that is only dependant in the weights of the network.Therefore, this function is optimizable with respect to the weights of the network so that the loss value is 0 and therefore, the network satisfies the boundary and initial conditions.

In the current example, figure 8 shows in the left part, the domain points corresponding to the initial condition, whereas the points in the upper and lower part of the figure correspond to the points where the boundary conditions will be implemented.

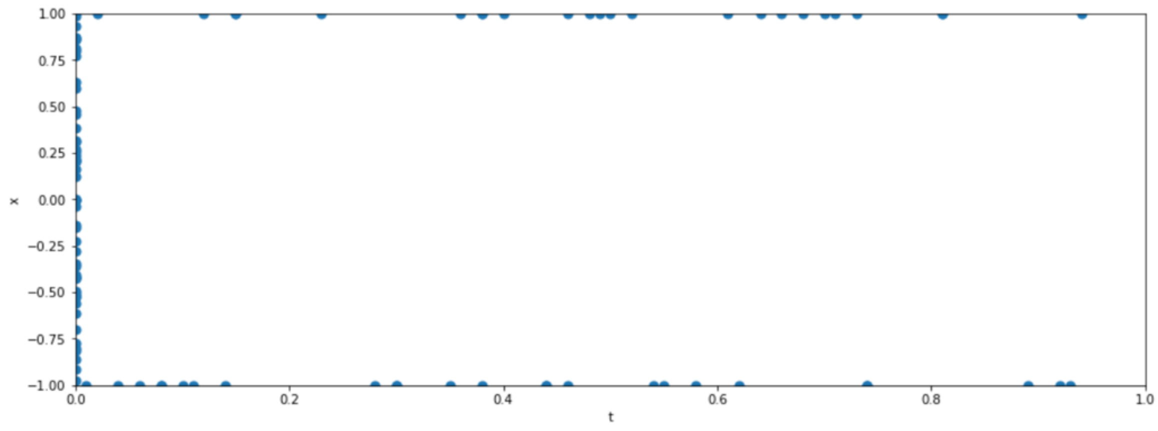


Figure 8: Training point set

The same process that has been carried out for the construction of the loss function that enforces the boundary conditions, is performed to enforce the PDE of the burgers equation, in this case;

$$f = u_t + uu_x - (0.01/\pi)u_{xx},$$

This expression will be evaluated in each of the 4000 collocation points represented in figure 9. It is important to note that differenciating the neural network with respect to different variables is done with the automatic differentiation toolbox that TensorFlow provides. This way, with the current set of weights, evaluating the network operator f in each collocation point, squaring it, and then computing the mean value among the 4000 points, we obtain a number that only depends in the network's parameters and represents in average, how separate is the network behaviour from the one that marks the burgers equation.

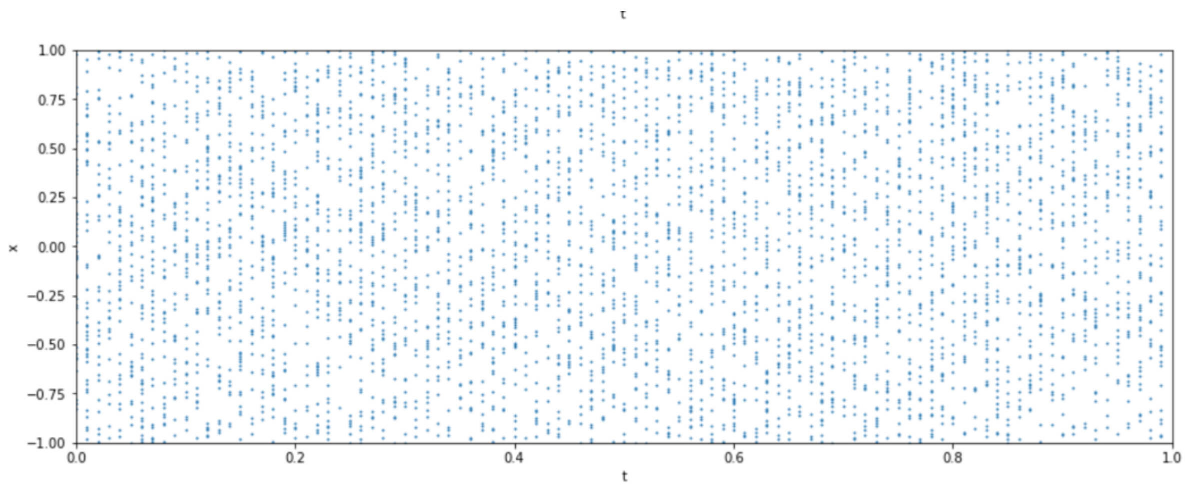


Figure 9: Collocation points set

Minimizing the sum of this scalar function with the one computed that enforces the boundary conditions with respect to the set of parameters the NN depends on, will deliver the set of weights whose network better approximates an analytical function that follows both the boundary conditions and the PDE, i.e the desired solution.

With this set up and data structure, after the optimization process of 6000 iterations (3000 with each optimizer) the loss value obtained is $L = 3x10^{-5}$ and the obtained result is represented in figure 10.

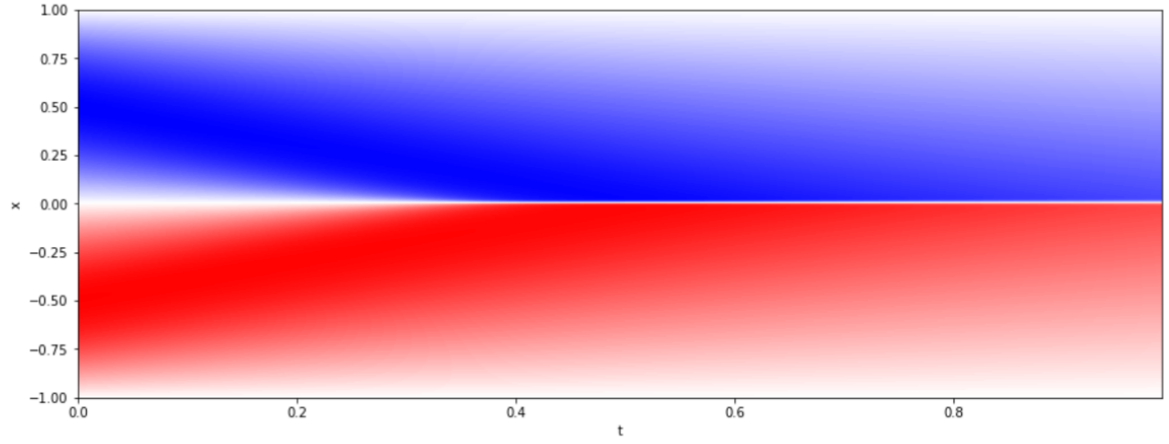


Figure 10: Simulation Results, Predicted solution

This result has been obtained in the following way; once the network has been trained until a sufficiently small loss value is achieved ([17],[6], [3] have considered sufficient loss values of 10^{-5}).

What we have obtained is an analytical function that approximates the solution of the problem we are trying to solve, this is, the set of weights has been updated (trained) so that the resulting function approximates as closely as possible the solution $u(x, t)$. The final step to represent this result in a figure is to predict the value of $u(x, t)$ in a grid of points that constitutes a discretization of the domain. Therefore obtaining a discrete representation of the PDE solution. This discrete set of points that is the solution of our problem is then represented by any of the multiple means that python provides. The relative L_2 error obtained in this simulation was of $L_2 = 5 \times 10^{-2}$.

In [17] , an error systematic study was carried out to quantify the predictive accuracy of this method for different number of Collocation and training points,as well as for different neural network architectures. As explained before in section 7.2, this study will be reproduced in detail during the master project.

The general trend shows increasing prediction accuracy as the total number of training data is increased, given a sufficient number of collocation points. This observation highlights a key strength of pyhisics informed neural networks: by encoding the structure of the underlying physical law through the collocation points, one can obtain an accurate and data-efficient learning algorithm. The same error study was performed for different network architectures with 100 training points and 10000 collocation points. As expected, we observe that as the number of layers and neurons is increased (hence, the capacity of the neural network to approximate more complex functions, the predictive accuracy is increased.

From [1], using the same techniques with different parameters and Neural Network structure, the results obtained were:

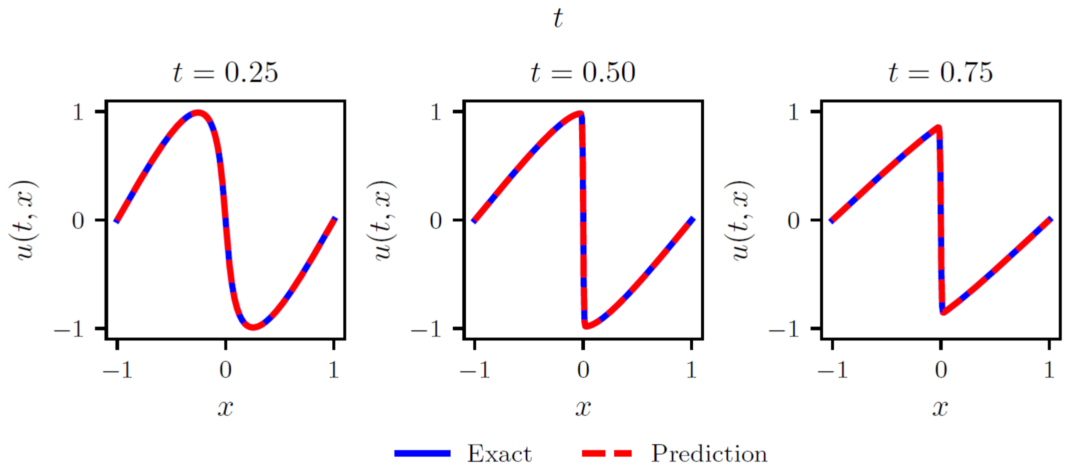


Figure 11: Comparison from [17] of the predicted and exact solutions corresponding to the three temporal snapshots, $L_2 = 6 \times 10^{-4}$

As an initial conclusion after this first simulation, one can determine that the the algorithm used is capable, to predict solutions with a fairly acceptable error ratio. Making it valid to other study cases and applications. Another quick observation that one can make is the simplicity of the whole algorithm with respect to classical methods. This fact also accentuates because of the use of TensorFlow as a machine learning tool. These two initial observations immediately pose PINNs as a simulation tool contender in the future, even as an academic tool.

8.2 Heat Equation 1D

To highlight the simplicity of the algorithm itself, we have computed the heat equation solution only changing the physical law. Boundary conditions, collocation points, training points and network complexity remain equal.

```
def f_model(X, model):
    u = model(X)
    u_x = tape.gradient(u, x)
    u_xx = tape.gradient(u_x, x)
    u_t = tape.gradient(u, t)
    f = u_t - u_xx
```

The prediction then becomes:

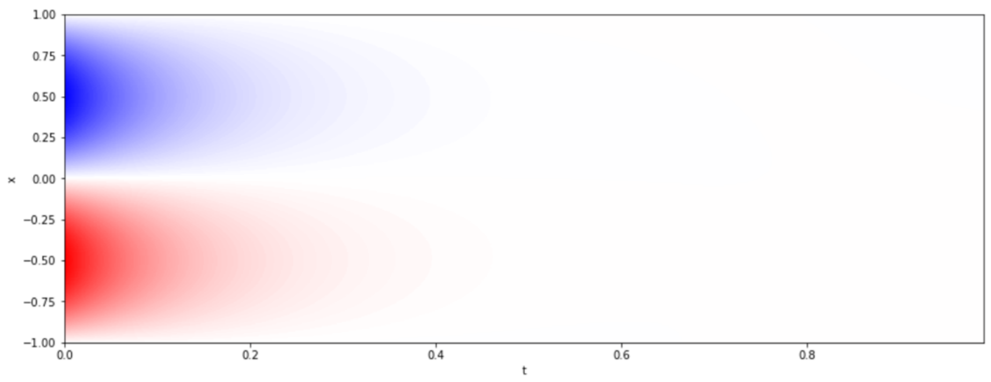
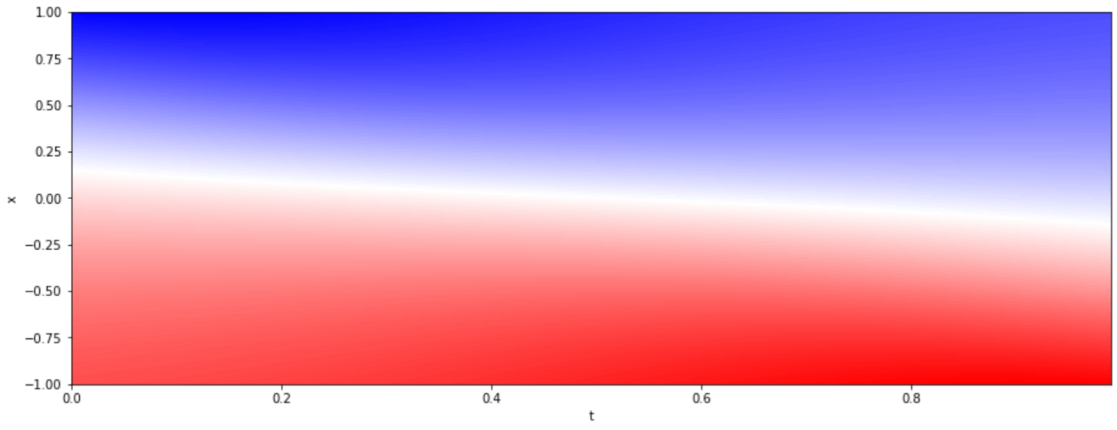


Figure 12: Heat equation prediction

In these two examples (the burgers equation and the heat equation 1D), the weights of the Neural Network have been initialized following a Glorot Normal Scheme, also called Xavier normal initializer. The way that is work is the following: the initializer draws samples from a truncated normal distribution centered on 0 with $\sigma = \sqrt{\frac{2}{fan_{in}+fan_{out}}}$ where fan_{in} is the number of input units in the weight tensor and fan_{out} is the number of output units in the weight tensor. With this scheme, for the heat equation problem, the initial prediction after initializing the weights may give a prediction like the following:



Obviously, the prediction fails to behave as the physical law determines basically because the network has not been optimized yet, but still, this result yields an interesting reflection; How important is the starting point of the network in terms of Convergence?

In both of this cases, the iteration process that optimizes the weights converges to a minimum of the cost function that correctly predicts the desired solution.During the master project, a deeper study will be carried out varying all the parameters the NN depends on and studying the evolution of the loss function value through the optimizing process.

Following this approach, we decided to see what part does the initialization of the NN play in the convergence of the weights towards the ones that predict the PDE solution. We have seen that parameters like the number of collocation points and the architecture complexity of the NN do tend to refine the solution predicted, but, how much time would it take to obtain this solution? Is it feasible? These kind of questions concerning the effect that the initial weights set, the neural Network architecture or the collocation points structure have on convergence have not been explicitly studied . During this master project we will, therefore try to answer them through different examples.

This aspect of PINNs is essential to consider it as a feasible method to solve PDEs in cases other than the academics that we have seen (e.g 3D Navier-Stokes equations). Otherwise if the model is not correctly tuned, convergence might appear after a sufficiently big number of steps, but the slow speed of convergence would prevent the method from being useful in practise.

The motivation of this study concerning the initial weight structure of the NN comes from the fact that, trying to study more complicated PDE cases, the network failed to converge giving predictions that did not correspond with the appropiate solution. After iterating through the parameters that constitute the algorithm, we concluded that the initial weight values of the network play a fundamental part in the algorithm. It is important to note that the method's convergence is not denied, but that it's speed under unadequate conditions invalidates it with respect to other numerical methods.

8.3 Poisson equation 2D

The next example that we will treat is the Poisson's equation in a 2D domain. Poisson's equation is an elliptic partial differential equation of broad utility in theoretical physics. For example, the solution to Poisson's equation is the potential field caused by a given electric charge or mass density distribution; with the potential field known, one can then calculate electrostatic or gravitational (force) field. It is a generalization of Laplace's equation, which is also frequently seen in physics. In our case, we will treat the Poisson solution as the steady state solution of the heat equation. The equation itself in the homogeneous case reads:

$$-\Delta u = 0$$

In the 2D case the equation is:

$$-u_{xx} - u_{yy} = 0$$

The domain in which we will solve the equation appears in figure 13 :

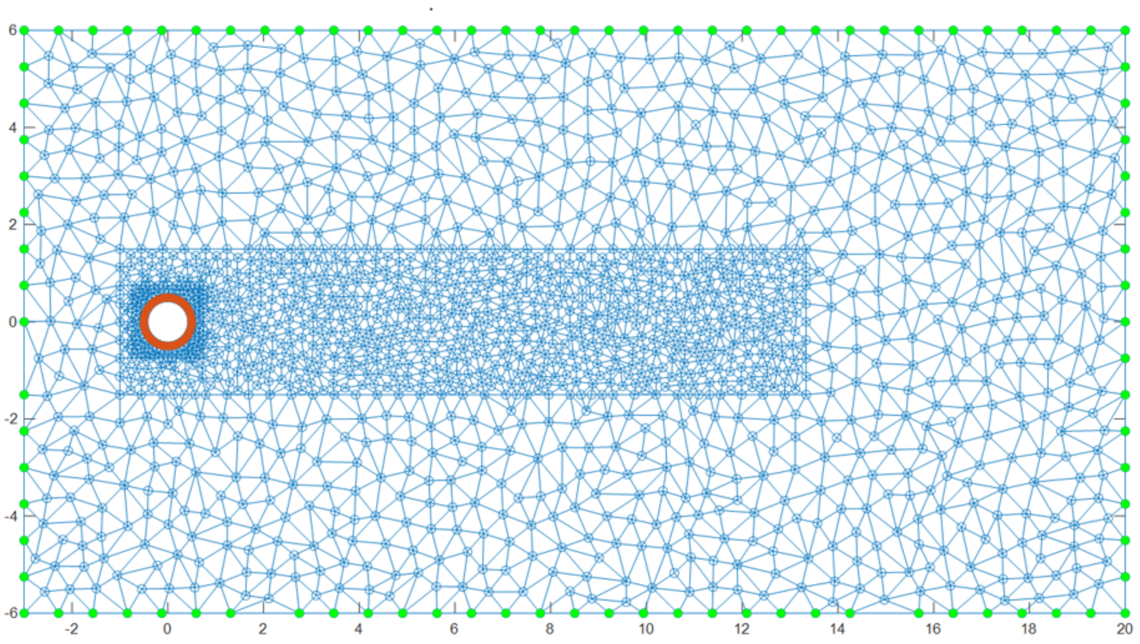


Figure 13: Heat equation domain

The boundary conditions are : $u = 1$ in the inner boundary (orange) and $u = 0$ on the outer boundary (green).There are several aspects about the previous mesh that important to emphasize. This mesh represents the collocation points which we will enforce $f := -u_{xx} - u_{yy}$. The green and orange dots are the training points in which the boundary values are enforced. In this case, there are 1719 collocation points and 125 training points.

The fact that we use this mesh responds to the necessity to have a structured set of points in the domain that we can work with, but the mesh itself is not necessary in the algorithm. The refinements seen in the area near the circle are there because the mesh was originally designed to solve the 2D Navier-Stokes equation. We will also use this mesh to represent the solution once the prediction is obtained. It is interesting to observe that almost 90% of the coding effort will consist on data structuring and formatting, the coding of the algorithm is fairly simple.

In this case, the number of inputs and outputs of the Neural network remains as in the burgers and heat equation that we have already worked with. After a similar iteration process, the obtained prediction is:

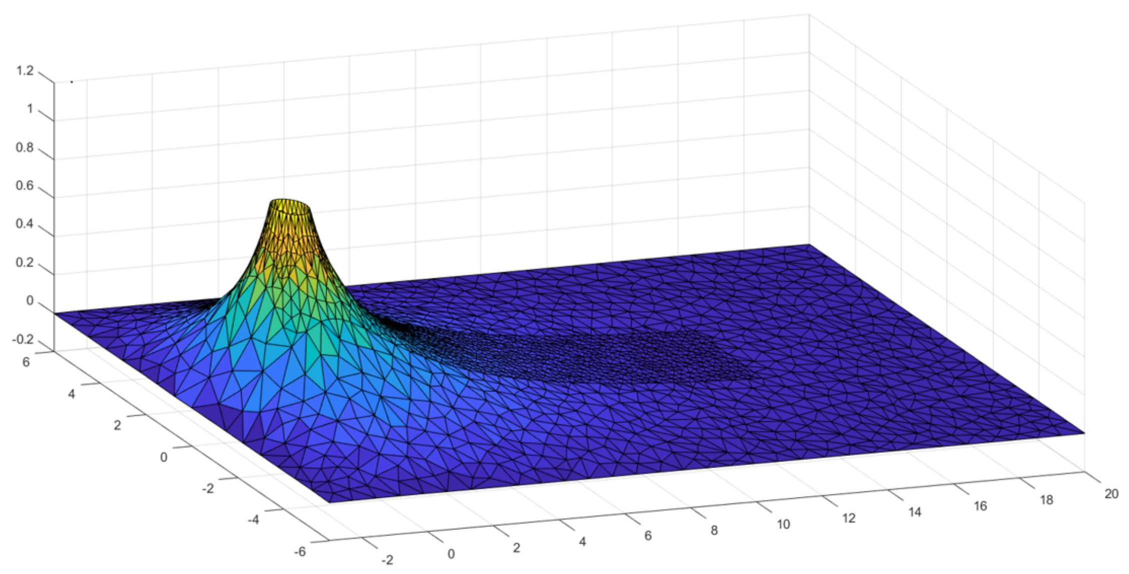
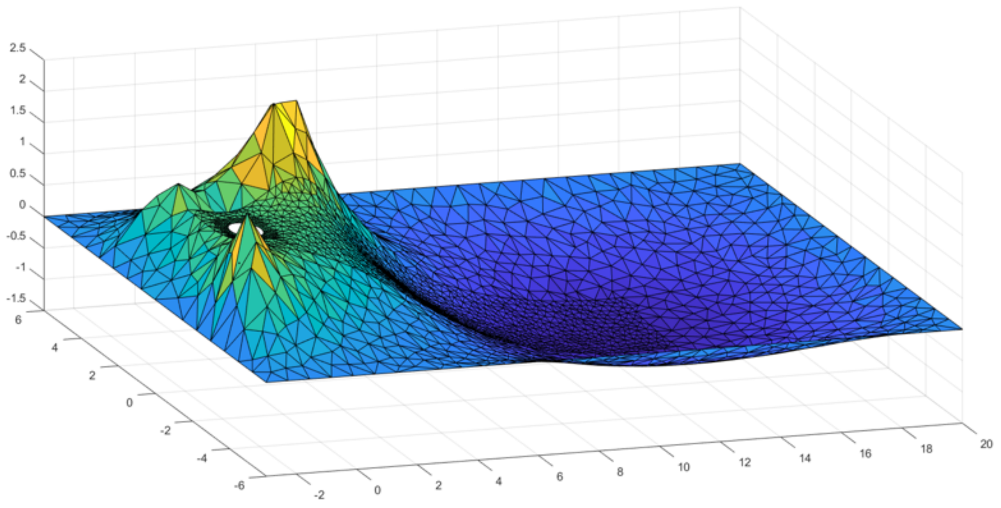


Figure 14: Poisson homogeneous solution

In this case, the Network had 5 hidden layers and 20 neurons per layer. A study of how simple can the network be still achieving convergence with an acceptable error is also due in the master project. As can be seen, the algorithm achieves convergence an an error study is applicable as in previous example. Trouble starts when the conditions of the problem change. We treat the Poisson problem with the same boundary conditions adding an inhomogeneus constant:

$$-u_{xx} - u_{yy} = -1$$

With the same problem conditions and set up, collocation points structure, optimization process, network architecture and initial weights, the prediction attained is:



The PINN algorithm fails to converge after a high number of iterations. It is important to note that this simulation has been carried out with a wide variety of different parameters that concern the network, i.e collocation point structure, number of training points or network architecture. All of them have turned out to be unsuccesful in achieving convergence. The fact that there exist a sufficient number of iterations in which convergence would be achieved is not denied, but the problem would become unfeasible under this solution method.

The only parameter left to change is the initial set of weights that define the initial shape of the network. Because the network architecture is analogous to the homogeneous case studied before, the weights set of the previous training is exported and then imported as an initial set of weights to this study case. Therefore, the network starts the training looking like figure 14. With this initial weight set, the obtained result is showed in figure 15.

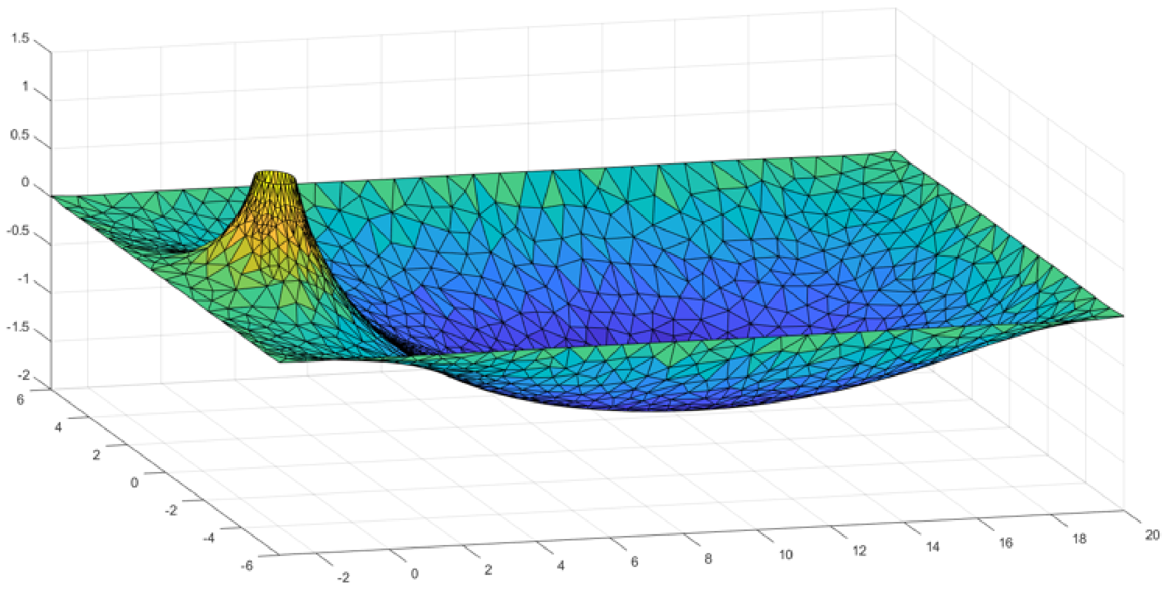


Figure 15: Succesful inhomogeneous Poisson simulation

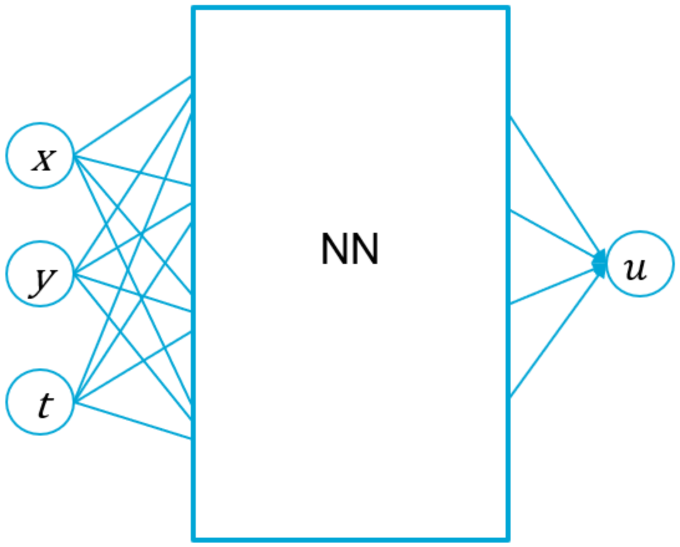
The simulation correctly converges towards the desired solution. Furthermore, it does in a very small number of iteration in comparison to the previous case. During the development of the master thesis one of the objectives is to work with the loss value evolution over epochs, but this evolution is undoubtedly faster in this last case.

Behind the reasoning of the fact that the optimization process works better if the starting point is well established comes from pure logic. The nearer you start from the solution, the faster you will converge. This fact results fundamental for complicated domains or complex PDE behaviour. As far as we have seen, the feasibility of PINNs as a PDE solver tool depends on its ability to converge in a reasonable time to the solution.

It has been therefore seen that features like the number of collocation points, the network complexity or the optimization process are secondary despite its effect in the error once convergence is achieved in this case.

8.4 Heat Equation 2D

The next problem encountered had to do with the initial weights structure for different networks.The heat equation in 2D needs a network with 3 inputs, and therefore, both the number of weights and the weight structure change, even for a network of the same dimensions. Since the number of weights isn't exportable from one network to the other, as soon as the network changes, the previous example isn't applicable anymore.



To solve the heat equation over the same domain, there is a new input; time, that changes the network architecture from having 2 inputs to 3. Therefore, even for the number of hidden layers and neurons per layer the weights structure is not shareable among the two problems.

We, therefore, must find a way to bypass this problem. The solution that is considered and posteriorly explained is to implement a helper loss function that, with stronger statements, efficiently sets the network weights in the desired initial position.

The collocation point structure considered is the following, using the same mesh as for the Poisson equation, we set a 3d mesh using the spatial mesh that we already had at different time instants. This is set up more clearly in figure 16.

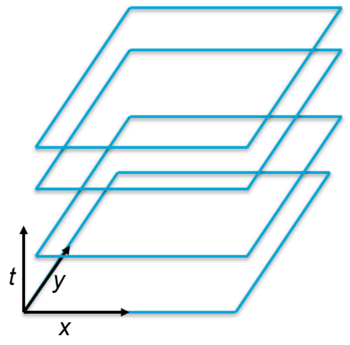


Figure 16: Collocation point structure for the heat equation

The set up of the equation in the domain has the same boundary conditions as the poisson case and the following initial condition seen in figure 17.

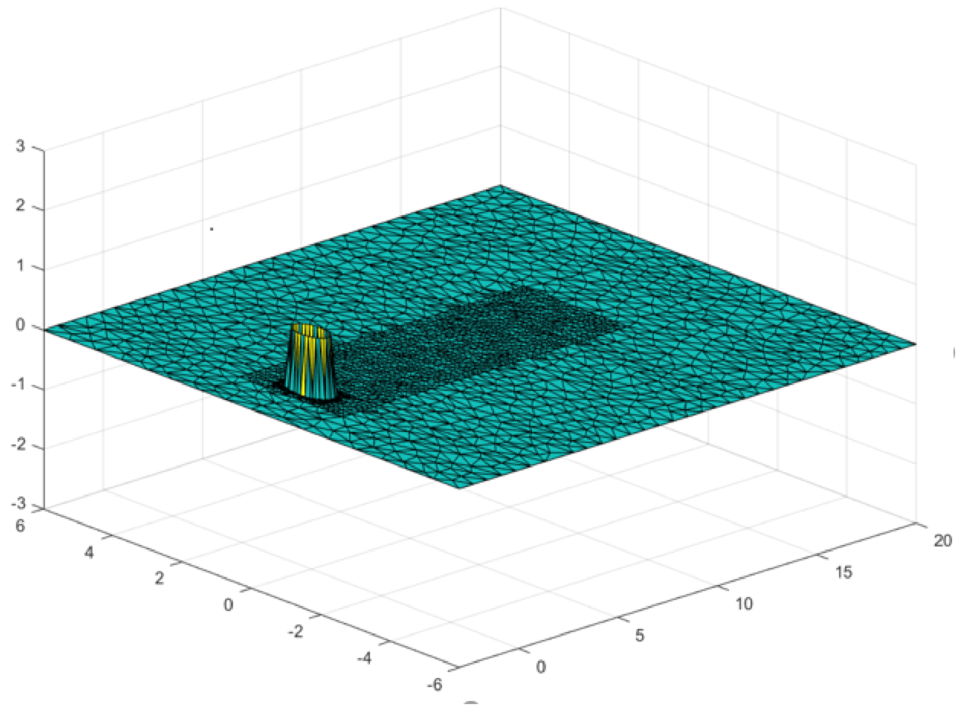


Figure 17: Initial condition for the Heat equation problem

Under these circumstances the expected result is an evolution from the initial condition set, to the steady state position provided by the Poisson equation that we had already solved.

This is one of the reasons why it was decided to start with the Heat equation, we know the expected behaviour to efficiently evaluate if the PINN has converged. Throughout the development of the master project a more thourough error analysis will be made with solutions computed by different numerical methods.

With this network architecture and parameter structure, the result in a certain time instant between the initial condition and the steady state solution, starting the mesh with glorot initial weights is seen in figure 18:

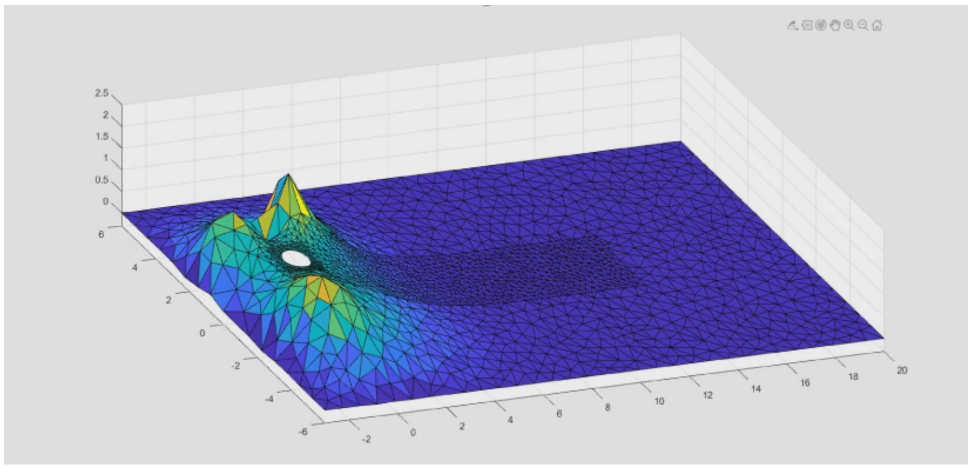


Figure 18: Intermediate time instant of the heat equation prediction

As can be seen, the prediction obtained does not converge. This set up was tried with a wide range of different network architectures, collocation point structure and magnitude, but even after 24 hours of training on the DAS5 super computer with 4 nodes, the results did not change. Therefore, we decided to change the set of initial weights in the following way.

Taking advantage of the fact that we have solved the steady state solution by the Poisson equation, we adjusted the set of weights so that the initial prediction,i.e shape, of the neural network would be the steady state solution in every time instant. This means that if we were to predict the solution of the heat equation without any training, the output solution in each time instant predicted would show the steady state result resultant from the Poisson equation that was computed in a network with a simpler architecture, i.e two inputs instead of three.

This is done via the helper loss function that will be optimized before the proper PINN optimization that was explained at the beginning of the document.

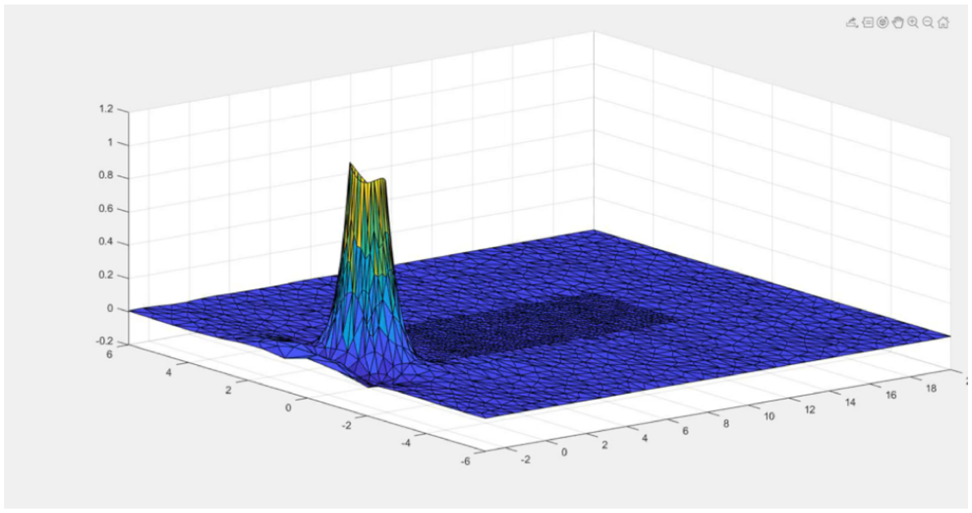
```
import tensorflow as tf

def helper_loss(u, X, model):
    u_pred = model(X)
    return tf.reduce_mean(tf.square(u_pred - u))
```

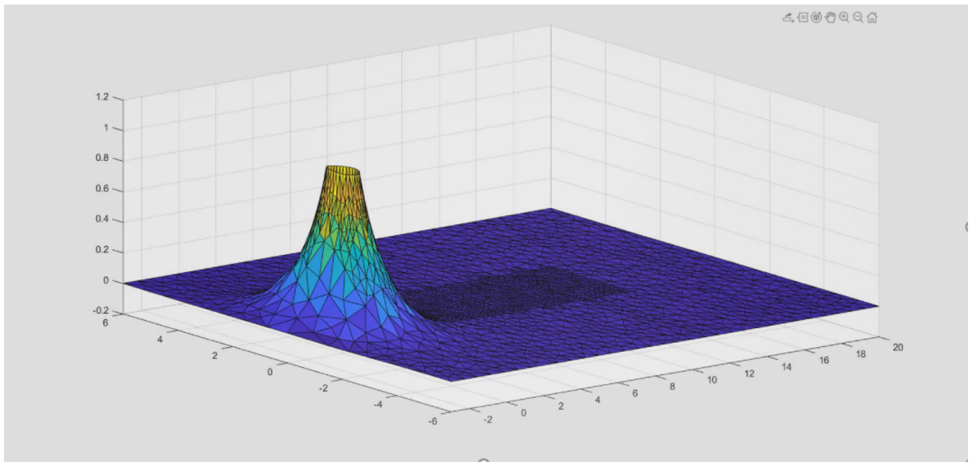
The u input vector is the one that specifies the wanted value in each point in the mesh. In this case:

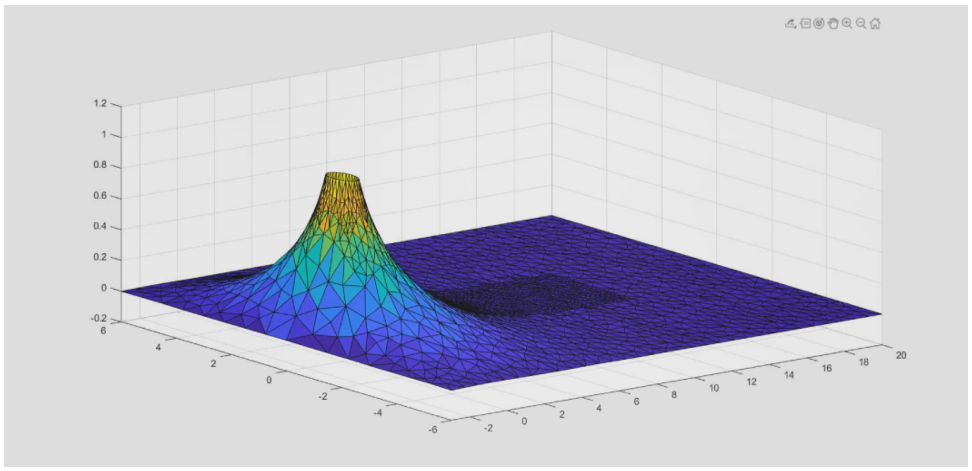
```
stat_pred = np.loadtxt('stationary_pred.dat')[:,None]

stationary_helper = np.tile(stat_pred, (N_time, 1))
```

The file *stationary_pred.data* is the one that contains the steady-state solution computed in the Poisson case. In this way, we continue with the comparison explained above that if the predicted values of two different networks look alike, it is probable that the weights do too.

With this final set up, the results obtained finally converge towards the correct solution of the heat equation. Below the prediction in different time instants is shown:







This confirms the hypothesis that the initial weight selection plays an important role in terms of convergence. Despite the fact that for a profound error study all parameters play a role, in the fundamental part of convergence a correct initial weight selection results necessary and highly determinant.

Without convergence, parameters like the number of collocation points, network complexity or the optimization process result secondary. Again this can be explained with the comparison that the more the objective function and the initial prediction without optimization look alike, the faster the optimization process towards convergence will be.

9 Parameter study, Introduction

As stated in section 8.1, this section intends to carry out a thorough analysis on the PINNs characteristics with respect to a number of parameters that the algorithm depends on.This will be done by combining different sets of values of these parameters. A set of error measurements will be evaluated in order to see which parameter structure optimizes convergence and error output. Related to convergence, it is important to note that the CPU time of each optimization process will be recorded in order to see the relation between error and computation time in each case.

One of the difficulties this study has, is the fact that the result obtained from a PINN depends in a relatively high number of parameters and it may be difficult to obtain a certain conclusion on the function that each of them offers. This way, the parameters considered for this study in the Burgers equation are the following:

- Number of collocation points
- Number of training (boundary) points
- Number of layers of the network
- Number of neurons per layer in the network
- Use of a helper loss function

  To pre-enforce some known prior data

  To pre-enforce the output result of another network (generally simpler)

As said, the effect of each of this parameters will be studied in order to know which one results more efficient under each circumstance. As a first approach, we have decided to select a criteria to reduce the number of parameters in order to keep track of the changes to achieve some relevant results. It is important to note the intention of this study is to be as thorough as possible in order to achieve significant conclusions. In order to reduce the number of parameters the PINN configuration depends on, the number of training points may be dependent on the number of collocation points corresponding to the following line of thought that is extracted from the curse of dimensionality, explained in [7].

There is an exponential increase in volume associated with adding extra dimensions to a mathematical space. For example, 100 evenly spaced sample points suffice to sample a unit interval (a "1-dimensional cube") with no more than $10^2 = 0.01$ distance between points; an equivalent sampling of a 2-dimensional unit cube with a lattice that has a spacing of $10^2 = 0.01$ between adjacent points would require 100^2 sample points, this is 100^d being d the number of dimensions. This effect is a combination of the combinatorics problems that deeply affect many applications of machine learning.

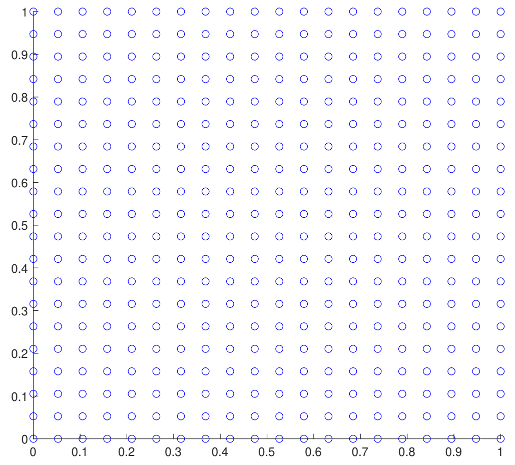


Figure 19: Collocation point structure for a 2D unit square with 400 collocation points

Applying this criteria,if we have a certain number N of collocation points distributed over a $2D$ square domain of side length L like the following case on figure 19, one can state that, in order to have a dimensionally congruent number of training points located in the boundary, the number of these boundary points must be $\sqrt{N}$. In the case of figure 19, the number of collocation points is 400 and the number of points dedicated to the boundary conditions enforcement is 20 so that the model is dimension congruent. This set of training points correspond to the points located in the boundary of figure 19. Therefore, in the following cases, the number of training points used in a 2D example can be defined as the square root of the number of collocation points defined for the problem. One issue that can be extracted from this criteria, is that the loss function correspondent to the enforcement of the boundary conditions is stronger, and therefore, less points are needed to make the prediction satisfy the B.C. In any case, the effect that these extra points have in the optimization time is considered to be negligible. Furthermore, it will be seen in later examples that the number of training points is not a critical parameter, neither in terms of computation time nor for the final error of the equation.

9.1 Possible uses of the helper loss function

The implementation of a helper loss function is implemented in two different aspects, depending on the prior data available. The first scenario encountered is going to be the case when some prior information about the solution is known. In case the solution of the PDE is known in a certain set of points, this can be used with two different objectives. The first one consists in evaluating the difference in the convergence rate between the optimization process with and without prior information, changing the amount of prior information enforced in the loss function. The envisioned hypothesis behind this proposal is that both the loss function and the error will have a faster convergence rate when a significant amount of prior information is available.

A possible function of having a complete set of prior information (the PDE solution) is to put a limit on the minimum error measure that we can achieve with a certain network architecture by training a network with the solution of the PDE itself.This way, we will obtain an error value solely limited by the approximation capacity of the network, and therefore, it marks the best possible solution that can be obtained independently on the rest of the parameters. The final L_2 error value obtained with this procedure will also be useful to evaluate how good a prediction is with respect to the one where prior information is available.

The helper loss function can be also useful to change the initial weight distribution of the network with data obtained via the optimization of a simpler network instead of starting the optimization with a glorot distribution on the weights. This distribution is by default initialized as explained in section 8.2. This approach intends to save computation time in the optimization process of more complex networks.

10 Error characterization on PINNs

Before explaining the mechanics of the parameter study that will be performed in the Burgers equation exposed on section 8.1 it is important to characterize the types of error that appear on PINNs as described in [11], [17] and [6].

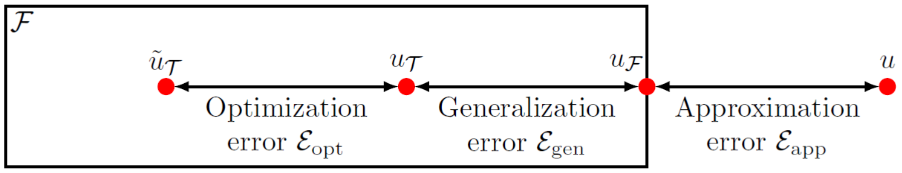

$\mathcal{F}$
$\tilde{u}_{\mathcal{T}}$
$u_{\mathcal{T}}$
$u_{\mathcal{F}}$
u
Optimization error $\mathcal{E}_{\text{opt}}$
Generalization error $\mathcal{E}_{\text{gen}}$
Approximation error $\mathcal{E}_{\text{app}}$

Figure 20: PINN error structure [11]

In [11] it is explained that Neural networks in practice have limited size. Let F denote the family of all the functions that can be represented by our chosen neural network architecture. The solution u of the PDE is unlikely to belong to the family F, and we define u_F as the function in F that is the closest to u. Because we only train the neural network on the training set T, we define u_T as the neural network whose loss is a global minimum. For simplicity, we assume that all these functions (u, u_F, u_T) are well defined and unique. Finding u_T minimizing the loss function is often computationally intractable [2], and the optimizer returns an approximate solution $\tilde{u}_T$. Then, we can decompose the total error ϵ as follows:

$$\mathcal{E} := \|\tilde{u}_{\mathcal{T}} - u\| \leq \underbrace{\|\tilde{u}_{\mathcal{T}} - u_{\mathcal{T}}\|}_{\mathcal{E}_{\text{opt}}} + \underbrace{\|u_{\mathcal{T}} - u_{\mathcal{F}}\|}_{\mathcal{E}_{\text{gen}}} + \underbrace{\|u_{\mathcal{F}} - u\|}_{\mathcal{E}_{\text{app}}}.$$

The approximation error, measures how closely u_F can approximate u. The generalization error is determined by the number and location of collocation points in T and the capacity of the family F. As explained, neural networks of larger size may have smaller approximation error but could lead to higher generalization errors, that could leads to difficulties in convergence. Finally, the optimization error stems from the loss function complexity and the optimization setup, such as learning rate and number of iterations.

It is important to note that, this generalization error, known in the supervised learning area by high variance on the predicted data, is the one responsible for the convergence problems that occur in cases like the one in figure 18. As stated before, once a generalization error settles in a prediction, it is very difficult to correct it. The loss value of the prediction remains very small while the resulting error is considerable.

Understanding these errors gives a deeper insight on the PINN characteristics and behaviour. As well as allow us to set a parameter study procedure, that will be mainly based on starting with a simple algorithm architecture, and step by step making it more complex being careful not construct and algorithm structure that would derive a big generalization error. This could lead to serious convergence problems which we try to avoid. Once the limits on how small can the error get modifying the parameter set defined, tools like the helper loss function will be used in order to concrete how useful they are in speeding the optimization procedure.

11 Parameter study procedure

Before explaining the procedure that will be followed for each case, it is important to remark the kind of information that we will get to evaluate whether the selected parameters are appropriate or not. From each optimization process we will receive:

- Evolution of the loss value w.r.t the number of iterations
- Final loss value
- Evolution of the L_2 error w.r.t the number of iterations
- Final L_2 error value
- Final absolute point-wise error
- Final physical point-wise loss value
- Optimization time

The word final, refers to the moment on which the iteration process is stopped, obtaining the data mentioned in the previous list. The errors that will be showed during the simulations, compare the obtained prediction with a point-wise solution from [17] considered to be exact. This way, for each optimization process we obtain different measurements of error that will help us characterize the obtained prediction. The only parameter that may appear to be unclear from the previous list is the final physical point-wise loss value. What this parameter evaluates at each point where a prediction has been made (once the optimization has finished), is the value of the function:

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

which, as described in section 7 enforces the PDE in each point where it is evaluated. This error measurement should be 0 in every point in case we had predicted the exact solution on the problem. It is important to remark that, by point-wise representing the value of this function over the predicted domain, we are "seeing" more information than the one provided to the algorithm. The information provided to the loss function, is the mean of the square value of f over the collocation points in the domain. The optimizer does not distinguish points by their individual magnitude of the error, and optimizes all of them as if the error was uniform throughout all the domain. Carefully looking at this error measurement will indicate the troubling areas of the solution we are trying to predict.

Finally, the last output data that will be evaluated is the processing time during optimization. This value will reflect the difference between different sets of collocation points and network architecture, and this way, we will be able to reflect which parameter set achieves the fastest a given error measurement. As a note, the optimization time of the program as it is designed namely represents its whole processing time.

Following this first analysis on the parameters we will use to evaluate the output prediction of each model, it is important to define the procedure on which this parameter tuning is going to be carried out. The intended technique consists in setting all parameters in the simplest possible way and escalate each of them individually observing the error fluctuations that the different parameters produce. The starting network architecture selected for the start of this study consists in a network with 2 hidden layers, each of them having 5 neurons. The reason for selecting this network architecture as a start, is because simpler network didn't offer sufficient approximation capacity, and the number of weights of the network is small enough for the optimization to be agile in the first cases. This way, we will compare processing times between this network and the more complex ones that will be evaluated.

12 Parameter study

The first step of this parameter study will be analogous for all the other cases that will be performed in this analysis. We will provide a lower bound on the error that the network configuration is capable to achieve. This will be done with knowledge on the prior information of the solution in the loss function. This is, knowing the exact solution of the problem, we will enforce via the helper loss function the exact solution of the problem. This methodology sets a lower bound on the minimum error achievable with a certain architecture.

In order to set some structure in the simulations that are being carried out, the following table indicates the value of the different parameters used in each optimization. Once all these parameters have been defined and showed in the correspondent table, the output evaluation parameters specified in section 11 will be displayed.

Collocation points	Training points	Layers	Neurons/layer	$\sharp$ weights
0**	4000	2	5	51

In this case, the number of collocation points that would normally enforce the physical law is 0 because we are enforcing the solution in the training points, that in this case are distributed throughout the whole domain. The number of weights in the table represents the number of trainable variables that the network has available to mimic the exact solution. This parameter gives good measure of the computational effort necessary to execute the optimization.

With the input parameters defined, we will perform the optimization process of the PINN and display the resulting output. As explained in section 11 these parameters constitute: solution predictions, optimization time, and error evolution.

The first output of this simulation can be seen in figure 21. This figure is analogous to figure 11 that was provided by [17]. We can observe the result from carrying out the procedure described before. The resulting prediction that takes into account the exact solution represents the maximum approximation capacity that the network has. Any result obtained with the classical PINN approach (only using information of the boundary conditions) with the same network architecture, will be only as good as the one in figure 21.

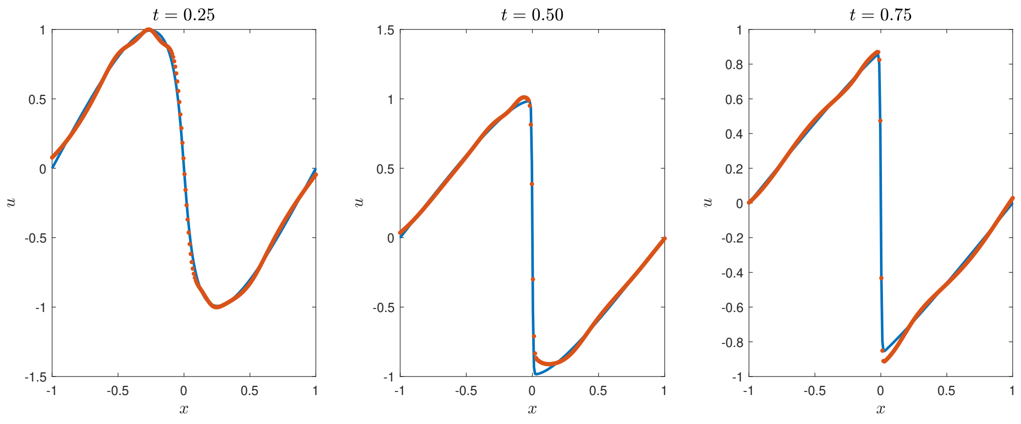


Figure 21: Predicted solution (orange) and exact solution (blue) in three different time instants

The error evolution on this optimization procedure can be observed in figure 22. This error evolution is represented with respect to the number of iterations made. In both cases, an error and loss measure was extracted after every 400 iterations. It is important to observe that both evolutions have been represented in logarithmic scale. We have considered to represent the error measures this way in order to better visually appreciate the convergence rate of the PINN in the loss function.

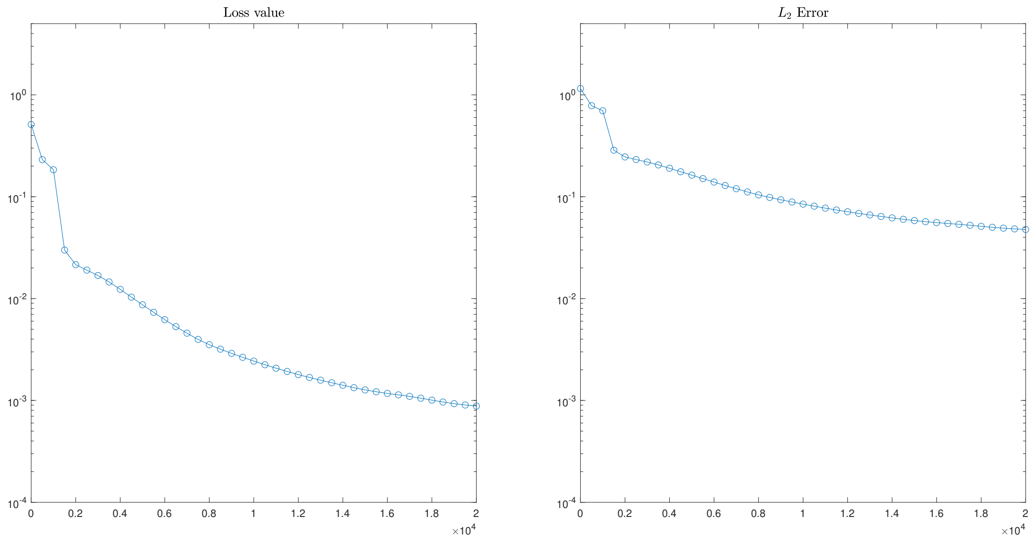


Figure 22: L_2 error and Loss evolution

We can observe in both graphs belonging to figure 22 that both error measurements stagnate after a certain number of iterations, indicating that the network has achieved it's maximum approximating capability. The final output evaluation parameters are shown in the table below. It is important to remark that, by doing this procedure of training the network with prior information before using the algorithm as an independent method, we are answering one of the first questions of the literature review, on which we express the possible utility of evaluating the flexibility of the network.

Final loss value	Final L_2 error	Optimization time
8×10^{-4}	5×10^{-2}	82 s

The main interest of showing the computation time on each case is to explore the possibility explained in the literature review of re-using the results obtained with simple networks.

These results would be obtained with less computational cost because the PDE is not being enforced and the cost of automatic differentiation is eliminated. The enforcement of the loss function is only via the exact solution.

The idea then is to train more complicated networks with obtained results from simpler networks (with a helper loss function). This way, we would be able to save a considerable amount of time. This first case illustrates this mechanic because the helper loss function is applied. Instead of using information obtained from another PINN, we are directly using the exact solution in order to analyze the approximation capacity of the network. It is very important to remark that the L_2 error has been studied in the points on which the network was not trained in order to get a faith-full result. This fact means that out of the 25600 points on which we know the exact solution, the network was trained to mimic a random subset of 5000 points and the error was evaluated in the other 20600.

12.1 Case Study 1

In this case, the first case study pursued will be the one with the following input parameters. As detailed in previous sections, these parameters signify the simplest of the cases, and as the results are obtained, we will increase the complexity of the algorithm in order to refine the results obtained.

Collocation points	Training points	Layers	Neurons/layer	$\sharp$ weights
200	200	2	5	51

The first observation that we can make before evaluating the output results of this iteration, is the fact that in this case the ratio between boundary points and collocation points is not the one specified in section 9. This is due to the fact that the number of collocation points resulted too small in previous simulations, the number of boundary points that would correspond was not sufficient to represent the boundary conditions. This lack of enforcement deeply hinders the convergence towards a desired solution. For this reason, it was decided to set a number of boundary points correspondent to a high number of the respective collocation points. This does not imply that the optimization cost will significantly increase since the loss function that enforces this boundary conditions does not have any automatic differentiation components. Again, that is why the training through a helper loss function results significantly less expensive.

The solution obtained after the optimization process in the three different time instants selected in [17] is:

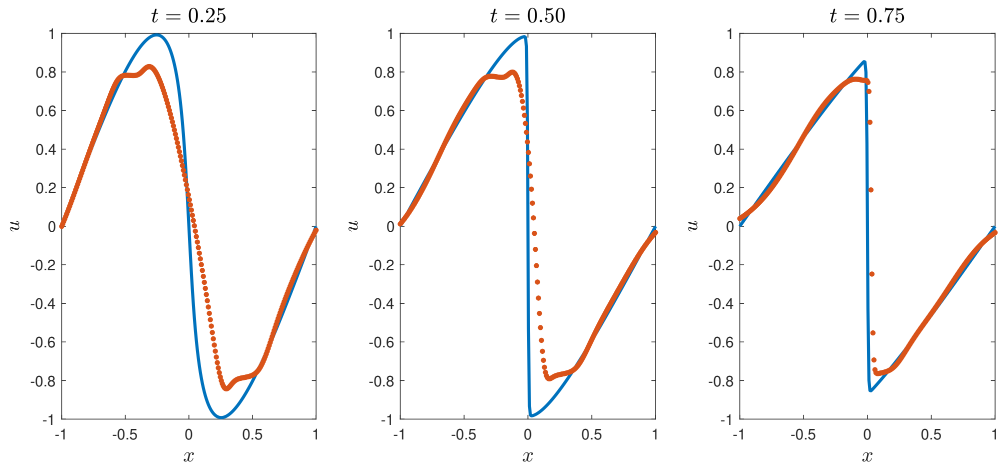


Figure 23: Predicted solution (orange) and exact solution (blue) in three different time instants

In figure 23 we observe the considerable difference if we compare it with figure 21. But even in this case, the predicted solution captures the esence of the Burgers equation behaviour, evolving from a sine wave to the rectilinear looking discontinuity seen during the last time instants of the simulation.

In order to get a sense on where the error lies, figure 24 shows the representation of the function f described in section 11. As explained, this function that was enforced in the loss function as the physical law describes in which points the physics of the problem is not being consistent. In figure 24 we see the absolute value of this function.

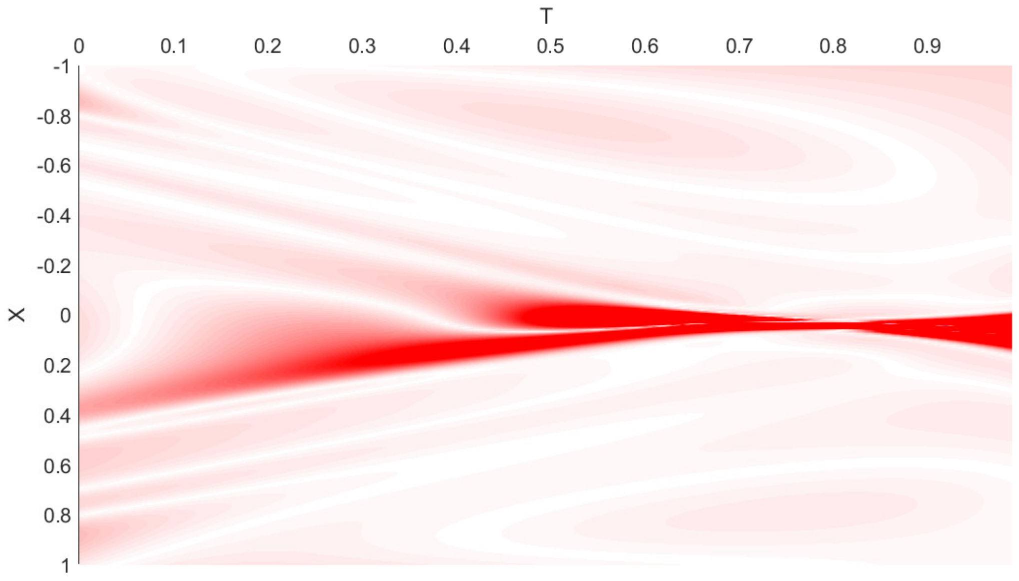


Figure 24: Physics enforcement error in the domain

We can appreciate a considerable spike in the absolute value of f in the area where the discontinuity is form. This physical error behaviour is logical since this discontinuity is known to be difficult to represent via an analytic expression. However, a very relevant conclusion that can be drawn from this figure, is that the error that appears at an early stage of the simulation in time, seems through the rest of the simulation with the physical behaviour set on the Burgers equation.

Therefore, figure 24 stresses the message that the initial conditions (In this example, part of the training points set) must be correctly and thoroughly enforced in order to get the best possible results. For this reason, even if the study cases have a sufficient number of collocation points to get a representative set of training points that are dimensionally congruent, as far as is does not deeply affect the computation time, we have decided to increase the training points a considerable level from this moment on-wards. This will prevent the propagation of this errors from the beginning in time on the simulation.

These two observations can be also appreciated in the absolute error evolution of the prediction with respect to the exact solution. In figure 25 we see this phenomena.

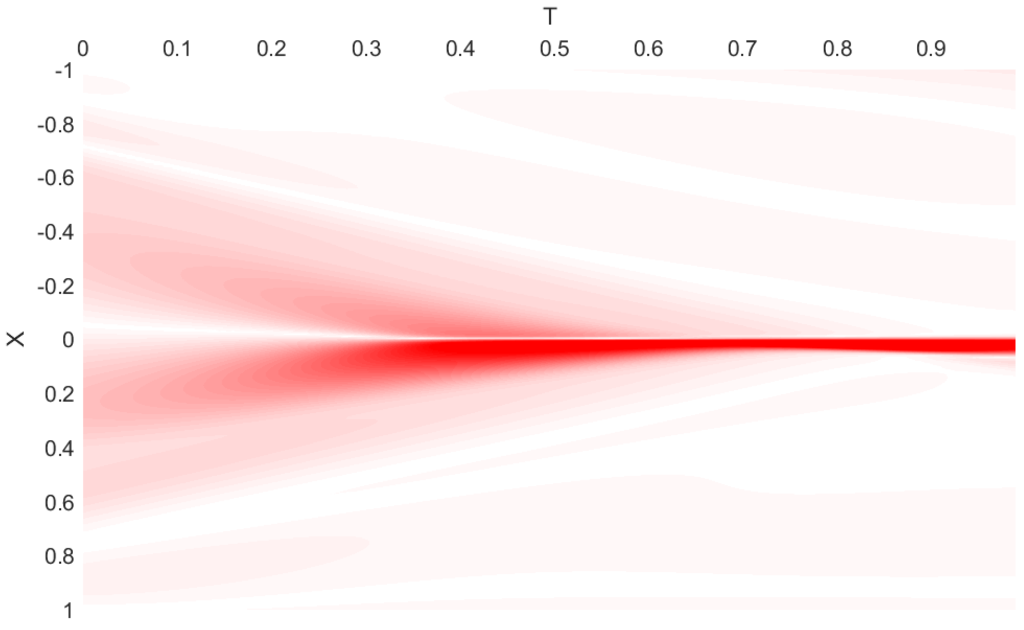


Figure 25: Absolute error in the domain with respect to the exact solution

It it important to take into account that this error propagation behaviour is applicable only to the burgers example. The behaviour of the equation together with the boundary conditions dictate that the evolution of the point in the position $(x_0, t_0) = (-0.5, 0)$ will be within a straight line that starts in that point whose slope is the value $u(x_0, t_0)$. This can be seen in figure 26 shown below:

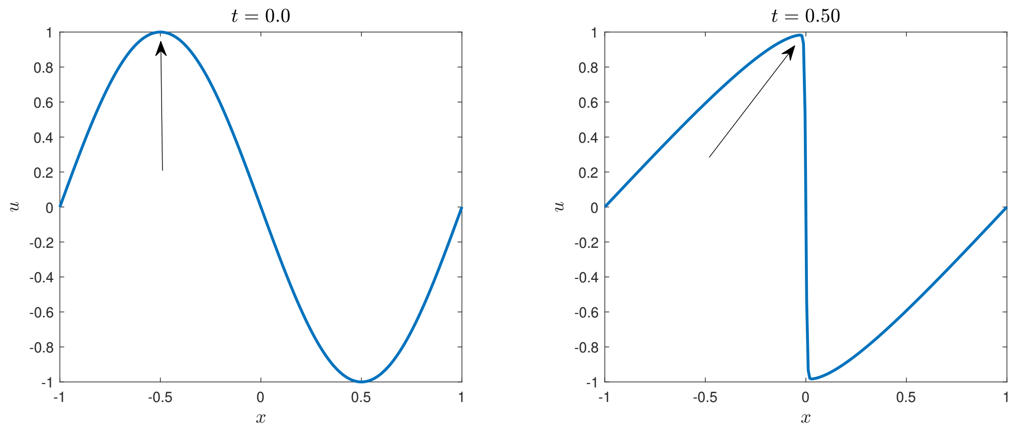


Figure 26: Exact Solution time evolution

In figure 26 we can see the evolution of the point indicated by the arrow. This point is displaced a $\Delta x = 0.5$ in a time interval of 0.5 seconds. This means that the relation between the spacial and time displacement is equal to 1, which is the value set by the boundary conditions:

$$u(x_0, t_0) = u(-0.5, 0) = 1$$

This can be appreciated in figure 24, which has been represented in figure 27 in a square format in order to appreciate the slope of the line described above:

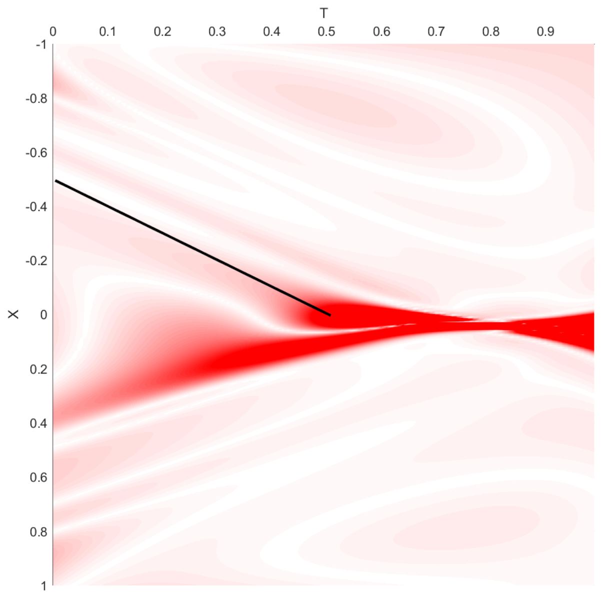


Figure 27: Error propagation through the physical equation

The error evolution of this study case can be seen in figure 28.

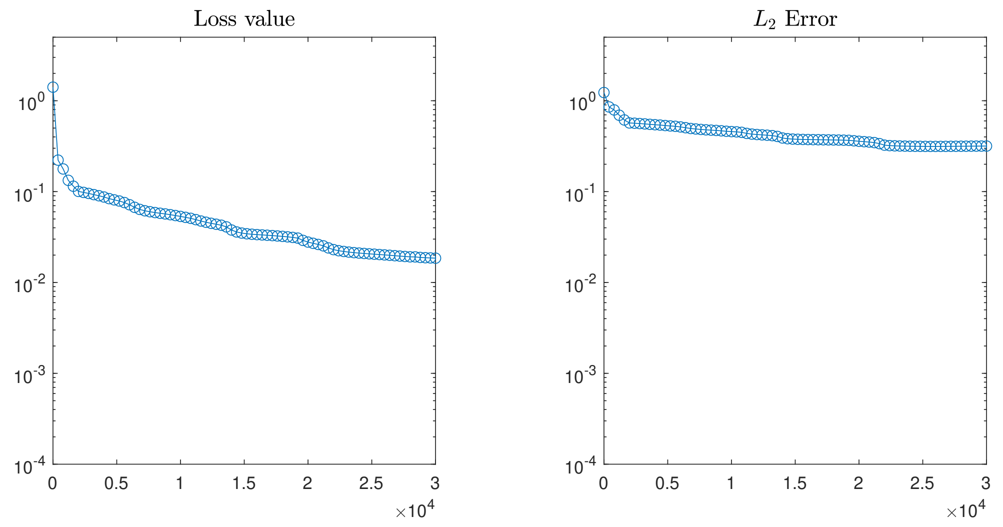


Figure 28: L_2 error and Loss evolution

The main observation that can be made from figure 28 is the slower convergence speed compared to the supervised training made in figure 22. In any case, this two convergence process are not comparable since the optimization technique is different. As logical, the optimization process stagnates on an error value that is considered to be the error that we are able to get with this parameters. The final loss value, L_2 error and optimization time are showed in the following table.

Final loss value	Final L_2 error	Optimization time
2×10^{-2}	3×10^{-1}	410 s

The optimization time significantly increases in this process, which is natural because in this case the network has to evaluate the PDE in each collocation point over each iteration. The final L_2 error is an order of magnitude bigger, which is a considerable error with respect to the exact solution. As explained during the parameter analysis, this is the intention. We are encapsulating the main source of error in the approximation capacity of the network. As we study this network architecture with different number of collocation points, we will see the little room for improvement this network offers, developing in each of the cases errors of the same magnitude. However, the main remark that we should extract in this case, is that with a very simple network and a small number of collocation points, the NN captures the main aspects of the solution, and this will be undoubtedly useful if we have convergence problems as we did in the Heat equation.

After the different cases study concerning the size of the collocation point set, we will represent a figure on which we will compare the evolution of the final error with respect to the number of collocation points in each case.

12.1.1 Collocation points evolution

The study that has been made in this section consists on iterating with a different number of collocation points maintaining the number of boundary points constant in every case. This number of training points is the one that would be considerably larger than the dimensionally congruent case to the maximum number of collocation points used in this study. This is, in the following table we can observe the different cases that have been performed.

Collocation points	Training points	Layers	Neurons/layer	♯ weights
200, 500, 1000 ,2000, 4000	200	2	5	51

For each of the 5 cases that can be inferred from the table above, we could extract a figure set as the one in section 12.1, but in this case, it was decided to only show the optimization on which 4000 points were taken into account. Once this results have been exposed, we will show the evolution of the L_2 error of the predicted solution with respect to the number of collocation points used in each case.

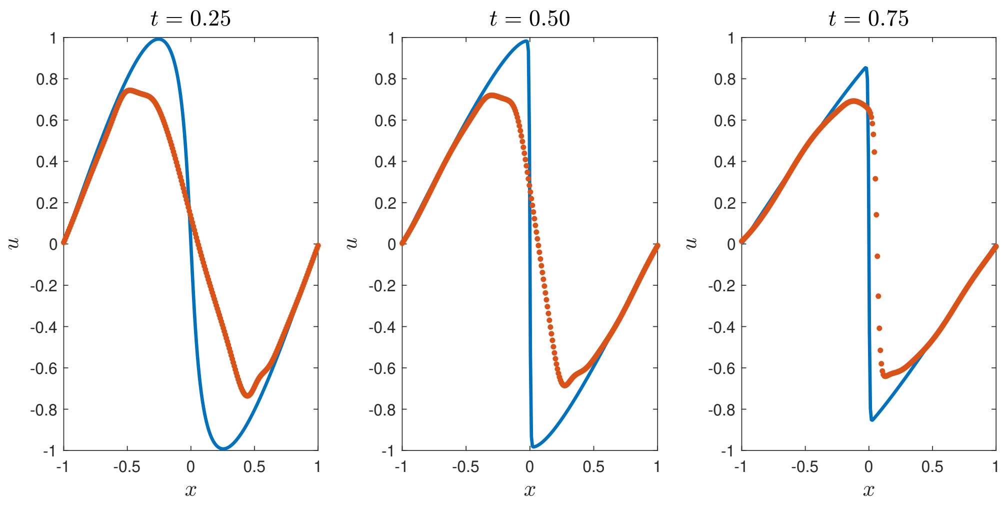


Figure 29: Predicted solution (orange) and exact solution (blue) in three different time instants

The result obtained is considerably worse than the one obtained in figure 23. This reflected in the convergence behaviour shown below in figure 30. The error evolution quickly stabilizes on a very high value that, despite the prediction approximately mimics the exact solution, it cannot be considered good enough.

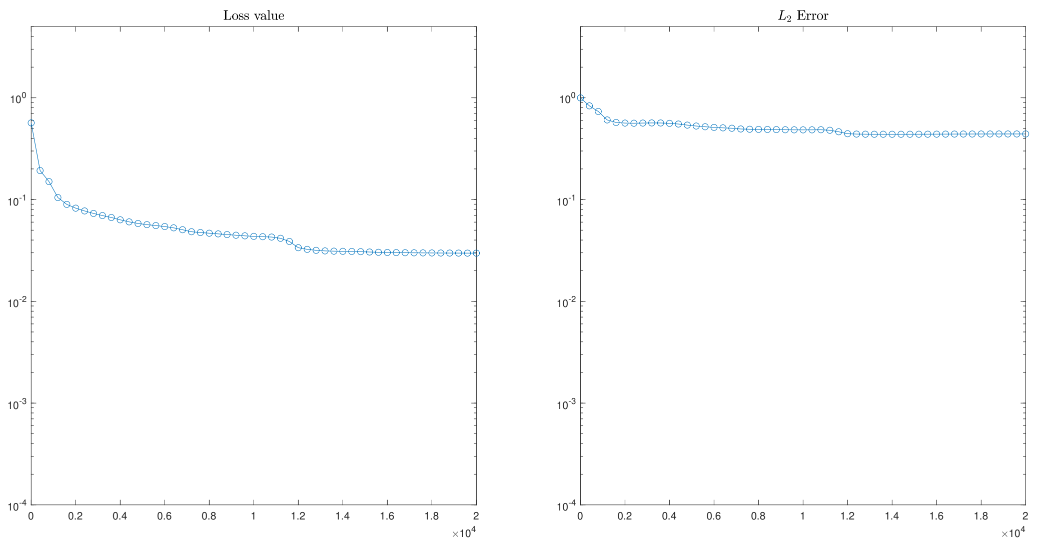


Figure 30

This behaviour is repeated when the collocation point set is decreased. The overall L_2 error stays constant in each different simulation made. This behaviour can be seen in figure 31 were the error with respect to the number of collocation points in the structure is represented. As remarked before, the rest of the parameters of the network stay the same.

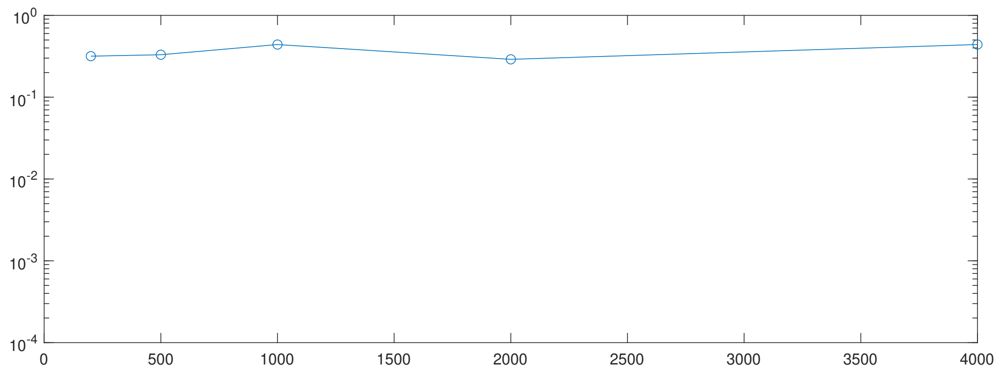


Figure 31: L_2 Error evolution with respect to the number of collocation points

There is no improvement in the error with the number of collocation points set. In this case, we have not kept increasing the number of collocation points because we know that with 4000 collocation points we will be able to get very good results as soon as the network complexity is increased.

The immediate conclusion that we can take is that the network is acting as a severe limiting factor on the error, up to the point where increasing the number of collocation points results counter productive, so that the network's prediction will not get any better but will undoubtely be more expensive. It has not been showed in a table like in previous examples, but the optimization of the 4000 collocation points set doubles the simplest one with 200 collocation points.

It then results relatively clear that the main source of error, and therefore the main parameter to change in the future, is the network architecture . However, before starting that study, the following analysis is going to be made. We are going to fix the number of collocation points to then change the ratio between these set and the training points, in order to see if this change can lead to an improvement of the error.

In any case, what really matters until now is the fact that we have captured the main features of the solution's characteristics with a fairly simple network and a small number of collocation points. These two are the parameters that slow the most the optimization process, and therefore the two main sources of possible time saving that we can achieve.

As explained, a final anaylisis before changing the network architecture will be carried out. The study of the ratio between the number of collocation and training points, allows us to set a starting dynamic with each PDE studied by a PINN technique.

In order not to show error evolutions that have a similar overall shape, the ouput of this ratio evolution will be showed in figure 32, that displays the L_2 error once this error function has stabilized during a sufficiently large number of iterations. This way, we will see an immediate relation between the ratio and the error for this network architecture.

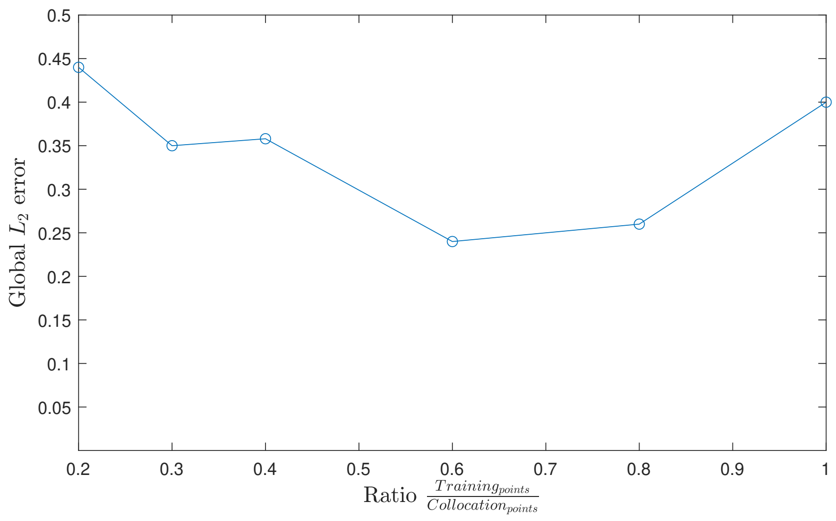


Figure 32: Error evolution w.r.t number of training points

There are a number of observations about figure 32 that are worth mentioning. While it is true that in this case, with this network architecture, the ratio between collocations and training points does have an effect in the final error (for $r = 0.6$ the error becomes half the error with other ratios), this final error does not change in the order of magnitude. This fact is important for two reasons, the first one is that, in algorithms of this kind, we are looking for error values of as small as 10^{-4}. The ratio does not become determinant in this matter. We will see in further analysis if this statement holds for more complicated networks. The second reason why it is important that the error does not change it's magnitude is, that with exactly the same input parameters, the error evolution might stagnate in values that differ with respect to one another, in this case, this difference can be up to 0.1. So, despite there is an effect on the ratio between these two sets of points, this effect does not become significant with respect to the change of other parameters in the algorithm.

12.2 Case study 2

Once the first case is concluded, the main conclusion that we can extract is that the bottleneck of the algorithm resided in the network architecture. For this reason, in this next case we will increase the number of neurons per layer of the network from 5 to 10, maintaining the rest of the parameters constant. Therefore, the table of input parameters becomes:

Collocation points	Training points	Layers	Neurons/layer	♯ weights
200	600	2	10	151

The first observation that we can make from the output results of this PINN case can be seen in figure 33. In this figure, we observe the evolution of the loss function and the L_2 error of this case study and the previous one, in which the number of neurons per layer (NPL) is equal to 5.

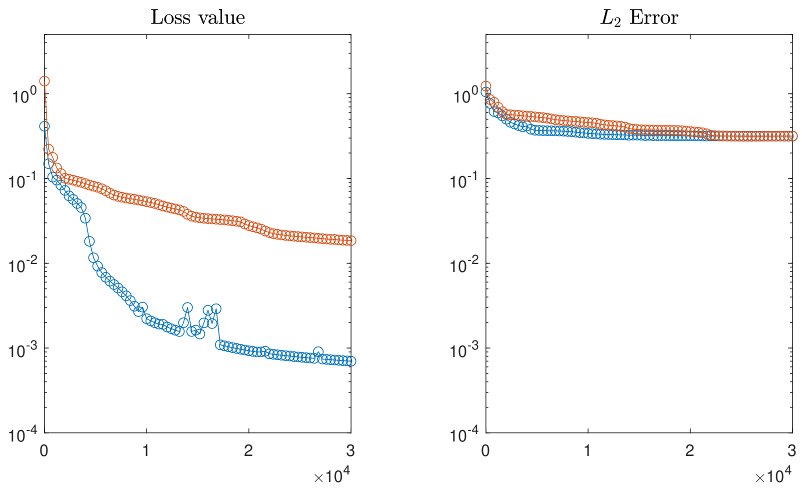


Figure 33: In orange, case 1 (5 NPL), in blue, case 2 (10 NPL)

The main observation that we can see is, that the evolution of the loss function evolves considerably faster towards 0 in the case where the network has 10 NPL. However, this evolution is different in the global L_2 error with respect to the exact solution.

This output result is very interesting because what it shows is that despite the fact that the more complex network has a much better approximation capacity (Loss figure), this is not reflected in the global L_2 error because in this case, the bottleneck of the algorithm resides in the number of collocation points. By doubling the number of NPL, the network triplicates the number of trainable parameters (151).

The output prediction of this first iteration is equivalent to the one showed in figure 23. There are still substantial mistakes despite the small value achieved in the loss function. Consequently, the next immediate step is to increase the number of collocation points and see how deeply this contributes to achieve a better error measure.

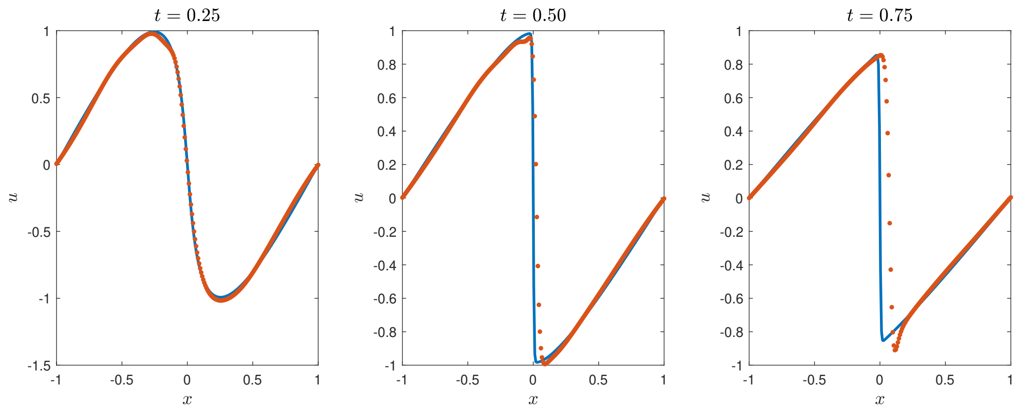


Figure 34: Predicted and exact solution in three different time instants

Then in the next case, the input parameter scenario selected is:

Collocation points	Training points	Layers	Neurons/layer	♯ weights
1000	600	2	10	151

The convergence result of this set of parameters can be observed in figure 35. It is important to note that it has been assumed that the number of training points is not decisive in terms of error and therefore we will only iterate with the number of collocation points.

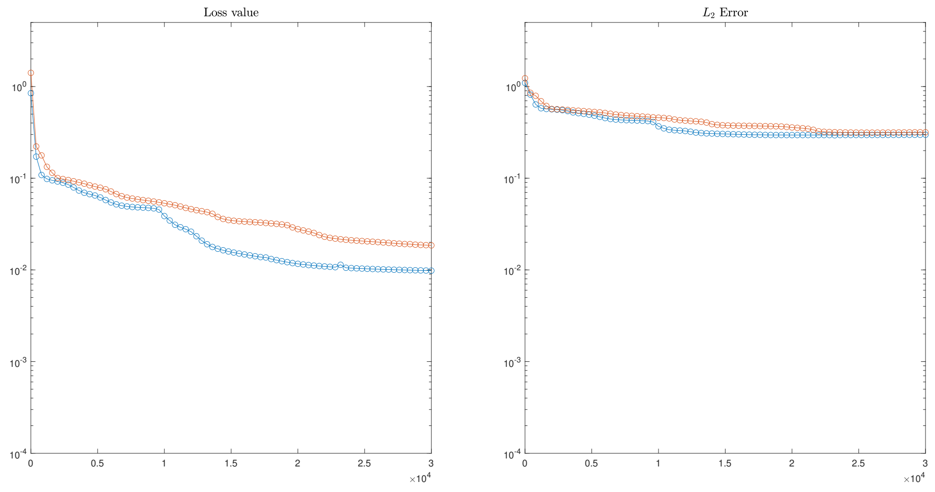


Figure 35: In orange, case 1 (5 NPL), in blue, case 2 (10 NPL)

Despite the number of collocation points has increased, there seems to exist no effect whatsoever in the error evolution. This is not the case of the loss value, in which we can observe a difference in its behaviour. There are a number of reasons why this could happen, but we will see that this lack of improvement in the error is simply because the number of collocation points was not sufficient. It is important to remark that these collocation points are evenly distributed throughout the whole domain with no refinement in any conflicting area.

Increasing the number of collocation points maintaining the rest of the parameters constant,the convergence results can be observed in figure 36. The parameters used therefore are:

Collocation points	Training points	Layers	Neurons/layer	♯ weights
4000	600	2	10	151

We have multiplied by 4 the number of collocation points and figure 36 displays the convergence results obtained. in this case, we do see results in the L_2 error. The orange pointed line seen in this figure corresponds to the error and loss evolution of a network with 5 NPL (and the same number of layers) trained with the same number of collocation points as the present study case. In other words, the only parameter that has changed between the 2 graphs represented in each figure is the number of neurons per layer in each of them.

A striking fact that is worth mentioning, is that the main difference between figure 36 and figure 35 is not the improvement of the L_2 error but the increased final value of the loss function. The difference (or the ratio) between the final loss value and the final L_2 error has increased, meaning that the final loss value and the final error are nearer to each other than they were in figure 35. It is then important to detect when a network is being infra-utilized with respect to the number of collocation points taken into account to enforce a given PDE.

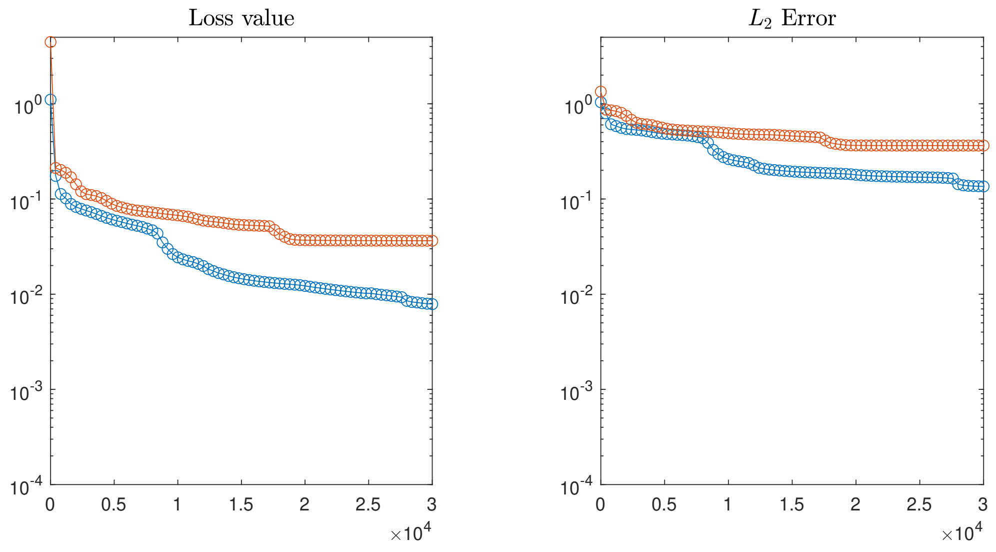


Figure 36: In orange, case 1 (5 NPL), in blue, case 2 (10 NPL)

Not doing so, would mean that we are "infra-utilizing" or network at the expense of the computational time used to optimize each of the parameters of this bigger network. Also, using too complex networks for the number of collocation points selected, increases the posibility of a high variance error that is very difficult to correct as it has been explained in section 10.

Consequently, we can state that this ratio between the loss value and the L_2 error is an output parameter that is able to indicate how well "tuned" the network is for the PINN we want to apply. Concretely, what we can say is that, if this ratio is too small, (big difference between loss function and error) then the network used is too complex for the point set selected and we must either increase the number of points, of decrease the complexity of the network.

Finally, for this set of input parameters, the output evaluation parameters are:

Final loss value	Final L_2 error	Optimization time
7×10^{-3}	1.3×10^{-1}	780 s

It is important to observe the notable increase in the computation time that this study takes. This increase in the computation time is due to the number of collocation points on which the algorithm evaluates the physical law. While in the kind of domains studied up till now this numbers are still manageable, as we have seen in section 9, as the number of input dimensions of the problem increases, it becomes more difficult to maintain this level of collocation point density inside the domain.

In order to set limits to the approximation capacity of each network, we will now finally represent the convergence evolution of the same network with 8000 collocation points. This is just to make sure that with 4000 collocation points we have reached the maximum approximation capacity and we can pursue more complex architectures.

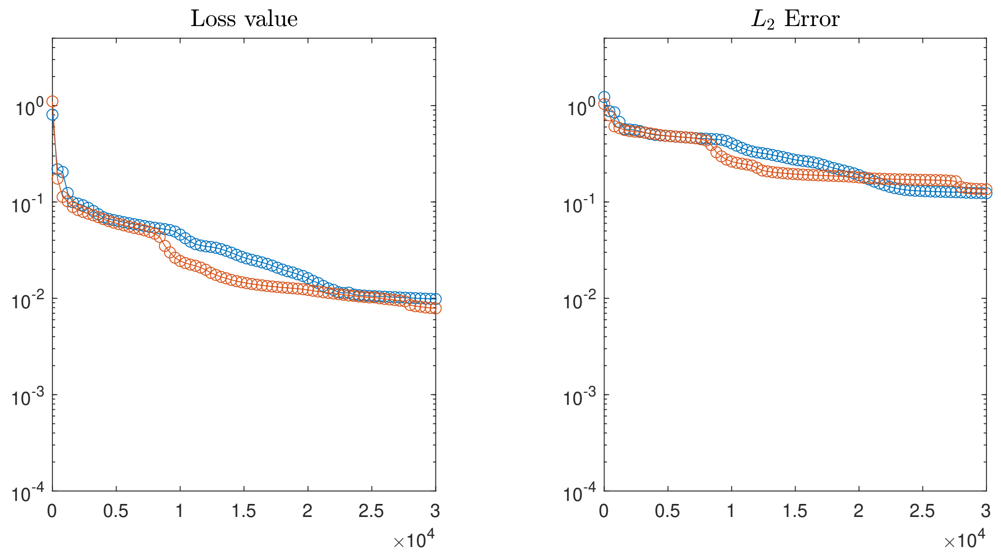


Figure 37: In orange, 4000 collocation points, in blue, 8000 collocation points.

Figure 37 shows that in both cases, the evolution of the loss function and the L_2 error follow a similar evolution. We can therefore confirm that the network has achieved it's maximum approximation capability. The only difference between the two results, is that the evolution with 8000 collocation points took a considerable amount of time to compute with respect to the 4000 collocation points case (1141 seconds). To obtain better results the solution inevitably goes through increasing the network complexity.

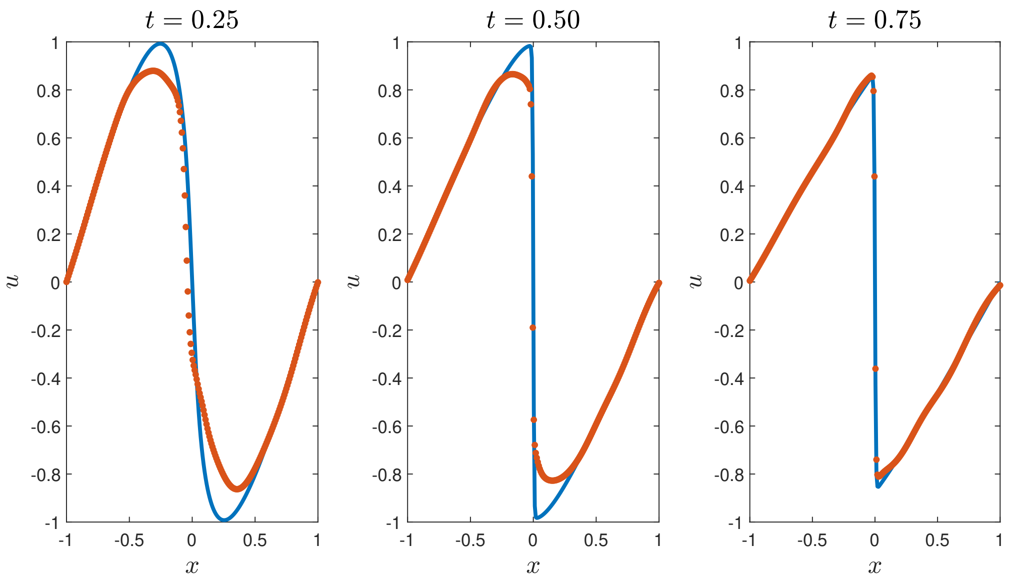


Figure 38: Predicted and exact solution in three different time instants (4000 collocation points, 2x10)

In this study, increasing the number of collocation points has also had an effect on the physics enforcement through the domain, as seen in figure 39.

.

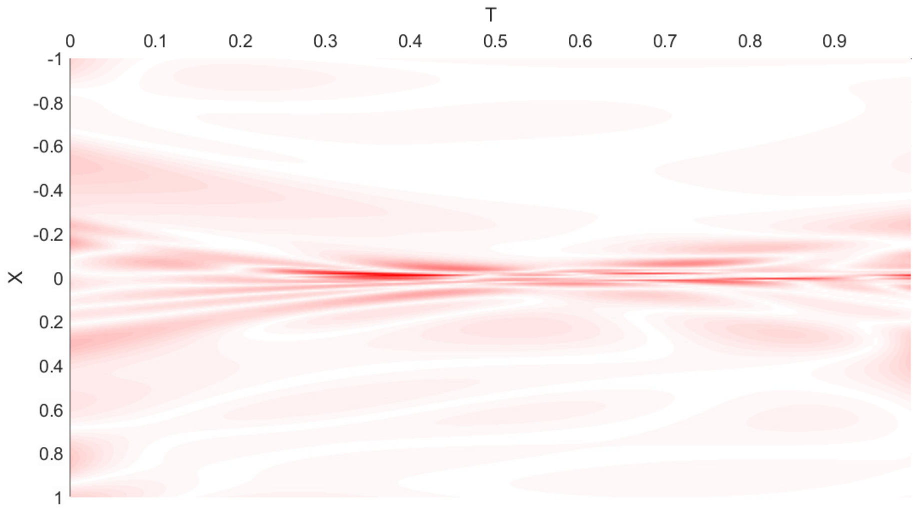


Figure 39: Physics error in the domain

We immediately observe that the physics error has plummeted with respect to figure 24 (the color axis has stayed constant).The absolute error represented in figure 40 is also consistent in this matter.

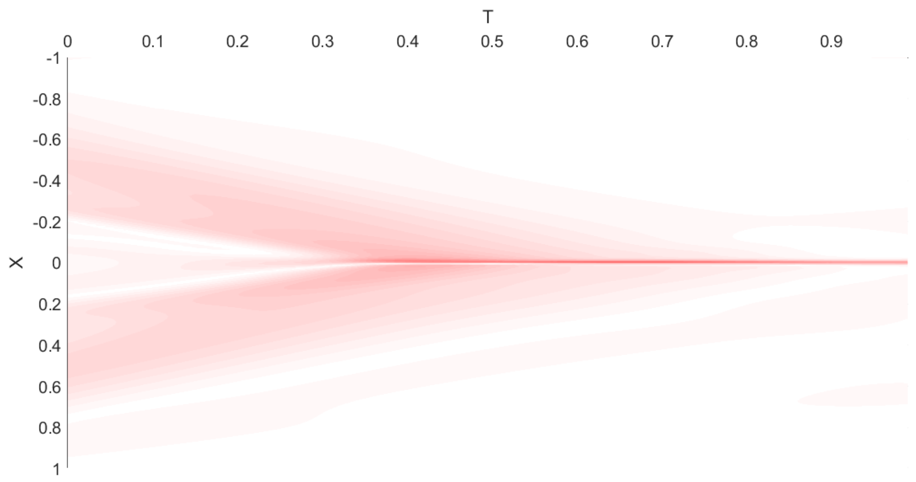


Figure 40: Absolute error in the domain

However, the error consistently stays in the area where the discontinuity of the burgers equation appears. It then seems logical, to increase the number of collocation points in the area where the network struggles the most to predict the solution.

In any case, before making a study on the collocation point refinement structure neccessary to decrease the level of error in the discontinuity, we have considered that it is worth finishing the structure study concerning the network architecture before pursuing this approach.

One immediate conclusion that we can take from this second case study is that in many cases in the literature, the networks are over-used , i.e, made too complex for the PINN application set for them. If this is done, we would be at a more complex case of the first example carried out in the case study 2, were the loss value quickly decreases while the L_2 error stagnates. This approach locates all the error due to the absence of collocation points, creating a lack of regularization in the network that could result in convergence difficulties in some cases. This error characterization has been more carefully explained in section 10.

Therefore, we will continue the foreseen approach, making the network more complex while making sure we achieve the ceiling of their approximating capacity. This way, we locate as much error as possible on the approximation limitation that the neural network has, assuring that convergence problems will not appear. The intention is to use very complex networks only to refine the results obtained before with simple networks and a small number of collocation points via the helper loss function. This way, we try to save computation time and take advantage of the whole approximating capacity of each network only when strictly necessary. In the next section, we proceed to increase the network complexity to observe it's effect on the error and loss evolution.

12.3 Case study 3

Once the second case is concluded, the main conclusion that we can extract is that the bottleneck of the algorithm resided in the network architecture. For this reason, in this next case we will increase the number of layers of the network from 2 to 4 instead of increasing the number of neurons per layer as already done in section 12.2.The number of collocation points selected in this case is the one that achived the maximum approximating capacity of the case study 2. i.e. 4000 collocation points. Therefore, the table of input parameters becomes:

Collocation points	Training points	Layers	Neurons/layer	♯ weights
4000	1000	4	10	371

As the number of trainable parameters of the network has substantially increased (371 with respect to the 151 from the previous network), what we can expect is a result analogous to the one obtained in figure 33. In order to be capable to observe if there has been any performance enhancement with this new set up, the loss and L_2 error evolution will be compared to the one obtained in the best scenario of the case study 2. This is, the one where we use 4000 collocation points. In order to clarify this, the following table compares the characteristics of the two input set of parameters studied. The color defined in the network number column acts as an indicator of the color of the respective convergence evolution figures.

Network number	Collocation points	Training points	Layers	Neurons/layer	♯ weights
1 (blue)	4000	1000	4	10	371
2 (orange)	4000	1000	2	10	151

This clarification is just with the intention of making the following of this report as comprehensible as possible. In any case, when the output prediction in three different time instants is displayed, the blue graphs will keep being the exact solution, which will be compared only to the results of the case study 3, as the comparison of the case study 2 has already been carried out in figure 38. In figure 41 we observe these two convergence evolutions.

The main conclusion that can be drawn from these figures is that the behaviour observed concides with the one in figure 38. Network 2 has more approximating capacity but this fact does not imply that the global L_2 error behaves the same way. Furthermore, despite network 1 doubles the number of trainable parameters, the L_2 error at the end of the optimization process is bigger in this network. As stated before, we can also observe that the ratio $\frac{Loss}{L_2}$ is very small $O(10^{-2})$ which we have concluded is an indicator of a lack of collocation points in the algorithm. In order to correctly tune network 1, we must increase the number of collocation points on which we evaluate the PDE.

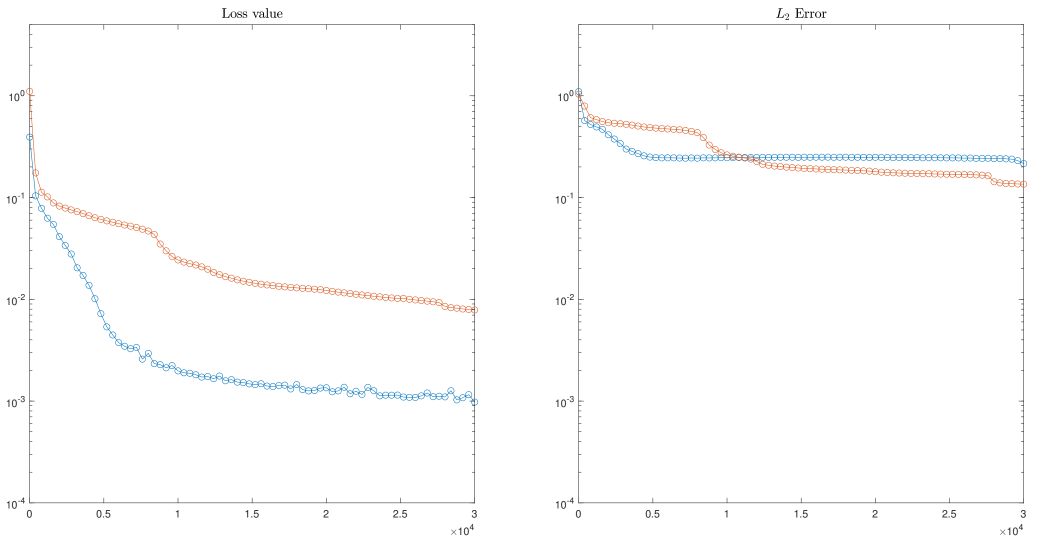


Figure 41: In orange, network 2, in blue, network 1. Both with 4000 collocation points

The ouput parameters of this network result:

Final loss value	Final L_2 error	Optimization time
9×10^{-4}	2×10^{-1}	1370 s

The result of the network not only gets a worse error measure, but also takes almost twice the time to compute compared with network 2. (1370 seconds compared to the 780 seconds of network 2). The input parameter set up not only is slower but also achieves worse performances than a correctly tuned PINN.

Just to deepen into the concept of the ratio between the loss value and the error, we are going to represent in the same logarithmic graph, both the error and the Loss evolution of this network. This way, it will be easier to observe the difference between the setup selected for this example with network 1, and the same set up increasing the number of collocation points.

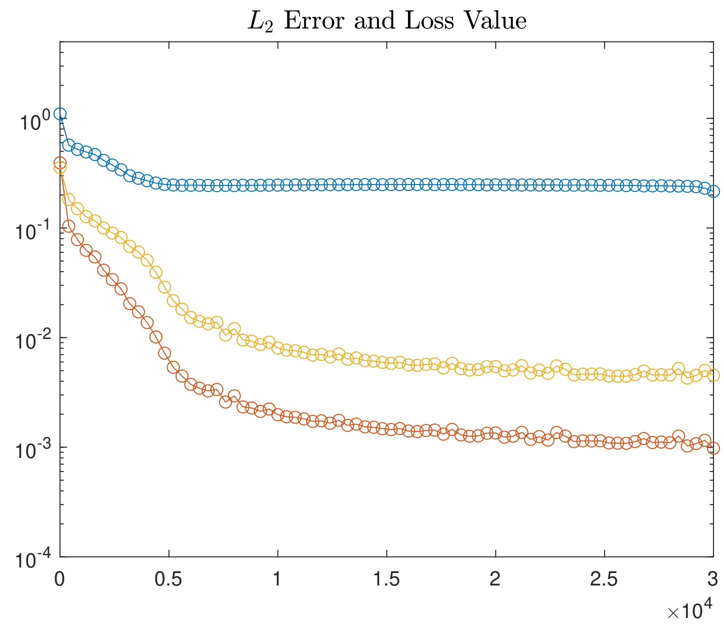


Figure 42: In orange loss value, in blue L_2 error, in yellow the ratio

It was considered to be visually more understandable if we showed the ratio defined as:

$$r = \frac{Loss}{L_2}$$

in each iteration of the process. This way it will be easier to identify when a network has not been correctly tuned by the previously expressed criteria. We will compare this result to the one obtained an increased quantity of collocation points.

Following the procedure carried out up till this step, we now evaluate the evolution of a network with 8000 collocation points maintaining constant the rest of input parameters. i.e:

Collocation points	Training points	Layers	Neurons/layer	♯ weights
8000	1000	4	10	371

Again, this evolution will be compared to the last well tuned example achieved, which is the case of network 2 (4000 collocation points 2x10 network). The convergence results can be seen in figure 654. Due to the performance enhancement that we will observe in this case, the results and the different error measures will be more thoroughly explained.

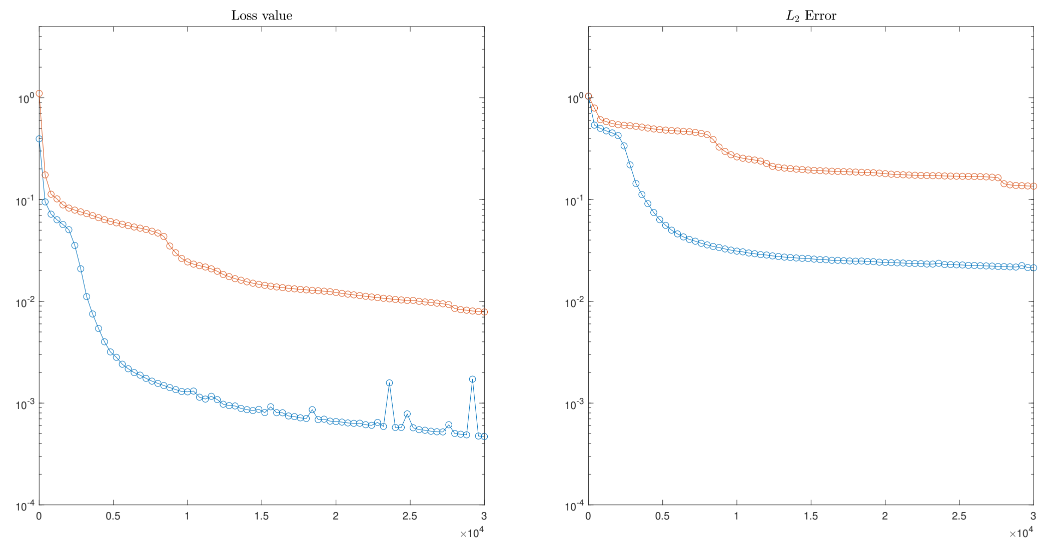


Figure 43: In orange Network 2, in blue case study 3

With the increase on the number of collocation points to 8000, the error evolution is corresponded to the loss evolution. In this case, we obtain a very consistent decrease on the error from the beginning, obtaining a final result of $L_2 = 2 \times 10^{-2}$.

The prediction of this network over the 3 time instants showed in previous examples can be observed in figure 44.

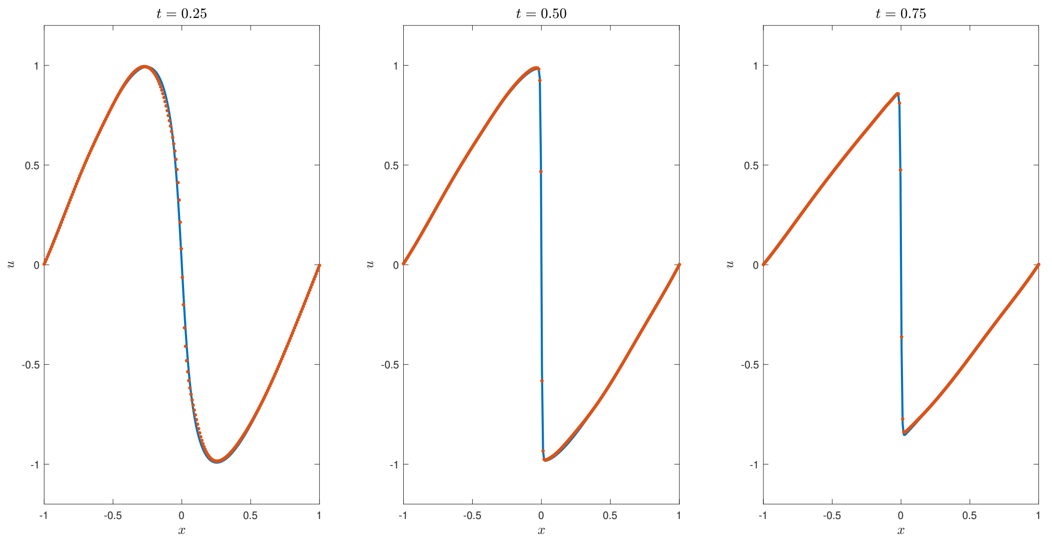


Figure 44: Prediction of the network with respect to the exact solution

The performance enhancement obtained from this network can also be observed in the absolute error over the domain in figure 45. We obtain high levels of refinement in every part of the domain.

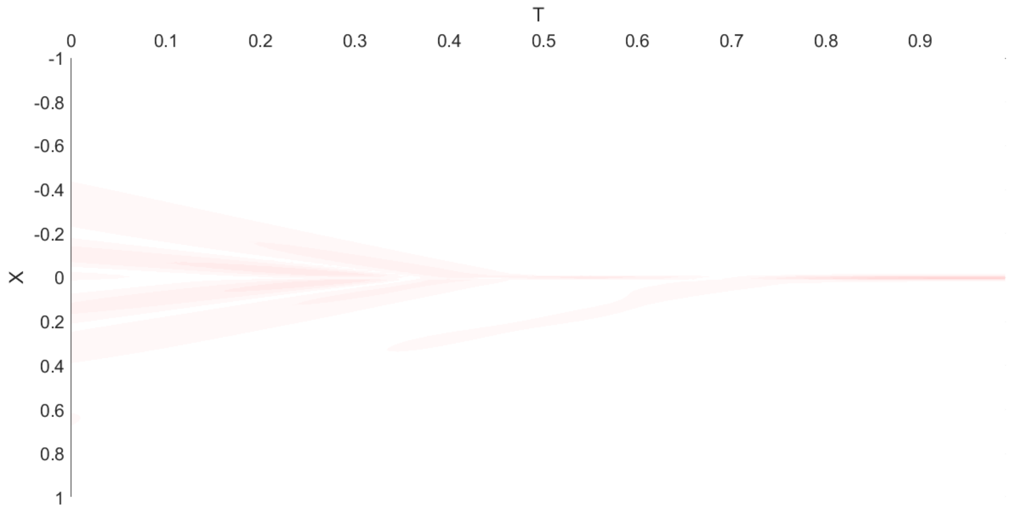


Figure 45: Absolute error with of the prediction with respect to the exact solution

It is important to insist in the fact that the color axis of this figure has stayed constant in every study case. Just to qualitatively compare the error, we can observe now figures 40 and 25 with respect to figure 45. The refinement has kept consistenly increasing as we were able to correctly tune the network towards better predictions. The physics loss once the optimization process stops is also a good indicator of the performance enhancement obtained in the network.

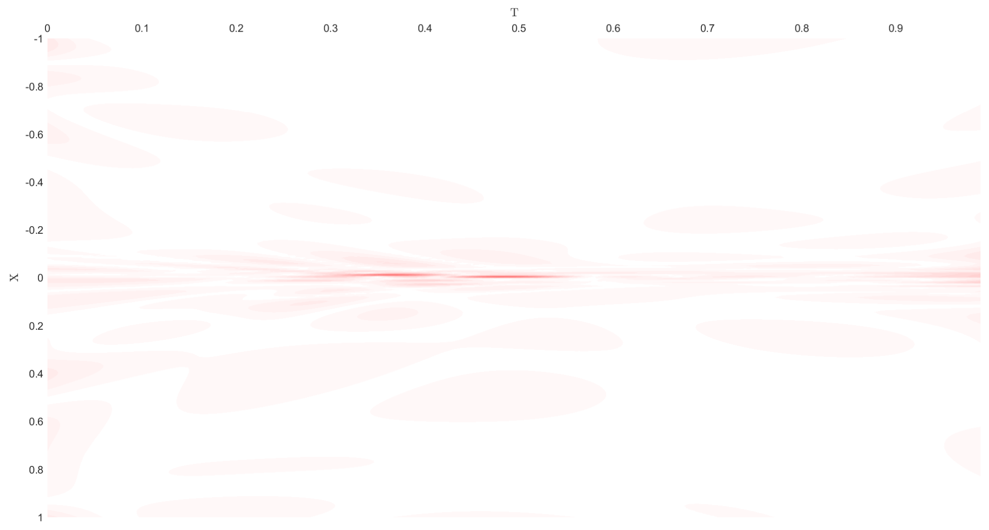


Figure 46: Physics loss distribution

It is very interesting to look in this case to the ratio between the loss value and the L_2 error. This is shown in figure 47.

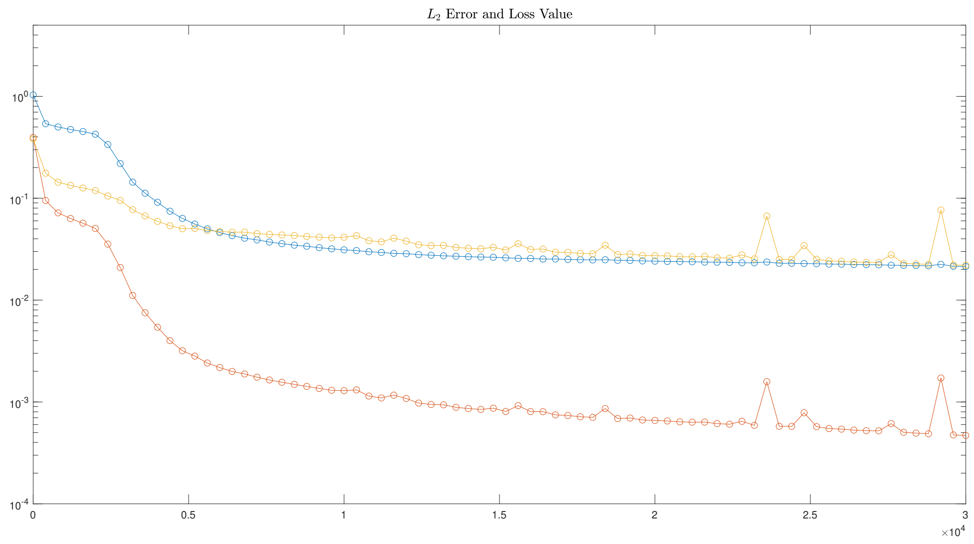


Figure 47

The ratio in this case is of the order of the L_2 error, which in this case is of the order of $O(10^{-2})$. The fact that both the ratio and the error are of the same order in this case is not necessarily an indicator of a certain property of the PINN set up. What we do think is that the value of the ratio is small enough for us to consider keep increasing the number of collocation points. But before displaying such a result, we are going to state the output parameters of this PINN set up.

Final loss value	Final L_2 error	Optimization time
4×10^{-4}	2×10^{-2}	1930 s

It is very important to note the significant increase of the computation time with the network structure and the number of collocation points. Obtaining a result with an error of this magnitude has taken slightly more than half an hour. While it is true that we may not have achieved the maximum approximation capacity of the network, time starts being a limiting factor. Not because of this example, but for more complex multi output PDE set ups. We have been able to achieve these results because the network only needs to deliver one output $u(x, t)$. If the problem to solve with the PINN has multiple output predictions, it may be interesting to keep track of the computation time more thoroughly.

Another important conclusion that we can take with respect to the simulations carried out in the literature [17], [15], [6], is that the network architecture taken for different levels of error, is significantly more complex than the one used in this proyect, achieving error levels comparable to the obtained in the literature mentioned. This fact is very important because the computation time is been showed to have a significant proportionality with the number of trainable parameter used in the network. The use of such complex networks has been directly translated in a waste of computational time in the examples shown in this papers. It is also important to state that, as mentioned in the error study of section 10, the use of too complex networks for the number of collocation points considered, could lead to high variance error inconsistencies that are very difficult to correct when the domain size and number of dimensions is increased (As studied in the heat equation of section 8.4).

Continuing with the same dynamic followed in previous study cases, we now study the the evolution of the error when the number of collocation points is increased up till 16000. The input parameter set for this case is seen in the table below.

Collocation points	Training points	Layers	Neurons/layer	♯ weights
16000	1000	4	10	371

The objective of the study of this set up is again, to set limits to the approximation capacity of the network. The convergence results can be observed in figure 48.

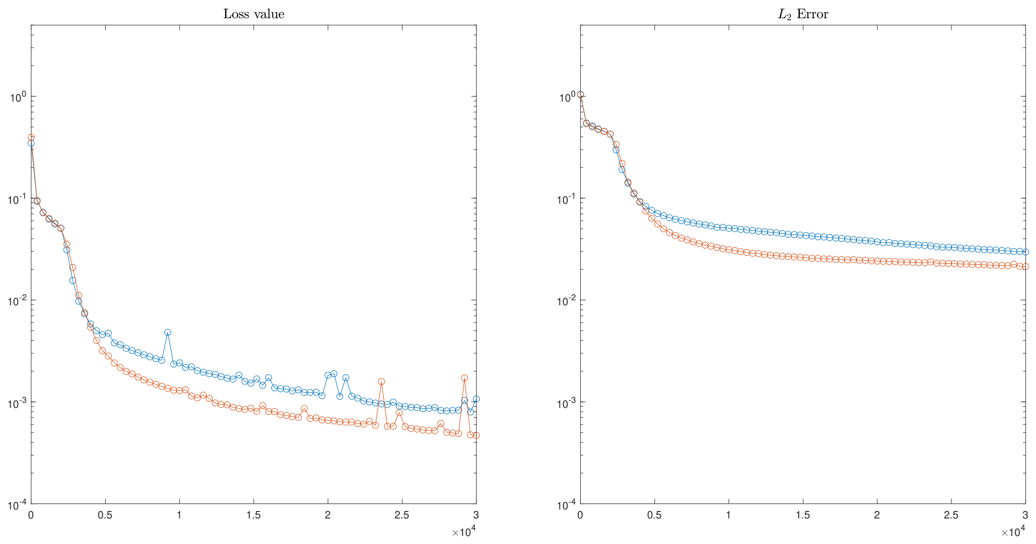


Figure 48: Convergence results of the PINN with 16000 collocation points (blue) with respect to the convergence result of the same network with 8000 collocation points

There are two main conclusions that we can extract from figure 48. The error and loss value obtained after the same number of iterations is roughly of the same order, but the error evolution of the 16000 collocation points case presents a negative slope at the end of the iteration process. Meaning that the approximation capacity of the network is not completely reached.

While this may be true, and if we kept optimizing the network we would be able to refine the results already obtained, the time that it would take makes makes the problem practically unfeasible. If the network does not have sufficient approximation capacity for the number of collocation points set, we will be slowing the method down because we are infra utilizing the refining capacity of the collocation point set.

The output parameters of this optimization are displayed in the following table

Final loss value	Final L_2 error	Optimization time
7×10^{-4}	3×10^{-2}	4965 s

Doubling the number of collocation points has multiplied the computation time by almost 3. While it is important not to scalate the network complexity too fast, it is also important to correctly select an adequate amount of collocation points, otherwise the PINN capacity to attain low levels of error will be prevented.

An important conclusion that can be drawn from this series of examples is that the tuning of the network is absolutely not trivial. There is a significant difference between a well tuned network and a network that has some parameter that is not consistent with the rest of the network. This difficulty makes relevant the person that carries out the simulation. If the input parameter set is not well tuned, the possibility that your PINN will be slow and dont achieve the best possible results is very high. This does not happen with other numerical methods.

12.3.1 Collocation point refinement

As explained in section 12.2, once we have achieved a sufficient approximation capacity, the motivation of this subsection is to evaluate whether a refinement of the collocation point distribution in the conflicting areas of the solution is useful in terms of error. For this mission, we will carry out a refinement in the collocation point distribution of our problem where we observe more error. By looking at figure 25 the conclusion is that, following this technique of adding more collocation points where the solution gives more error, the "red" areas will receive more collocation points.

For doing so, this study will be carried out in two cases, varying the number of total collocation points set in the domain. The overall number of collocation points including the ones added in the refinement in the two cases studied will be 4000 and 8000. These two numbers coincide to the cases studied in section 12.3. Consequently, we will be able to compare in each case, the error obtained with and without refinement, maintaining the rest of the parameters of the PINN constant. In all the previous cases, the collocation points have been randomly selected from a equally spaced mesh of 25600 points.

In this first study, the two cases that will be compared are showed in the following table. The only difference between them is that that network 1 has more collocation points on the area where more error was observed.

Network number	Refinement	Collocation points	Training points	Layers	Neurons/layer
3 (blue)	yes	4000	1000	4	10
4 (orange)	no	4000	1000	2	10

The collocation point structure chosen for network 1 can be observed in figure 49. From the 4000 collocation points of the mesh, 2120 are located in the area where the discontinuity of the exact solution has its presence. The reason for choosing only 4000 collocation points, is that in the uniformly distributed case, the network was fully capable of fulfilling the optimization of the physical law in the set of points, the approximation capacity of the network was not reached.

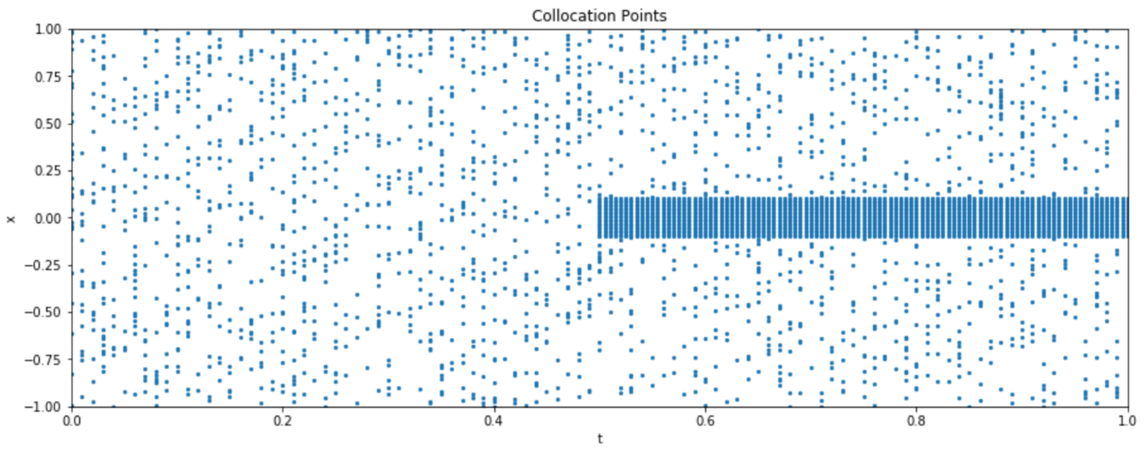


Figure 49: Collocation point distribution in the first refinement

The increase of the collocation point set in that area, tries to capture as much as possible the behaviour of the exact solution, shown in figure 50.

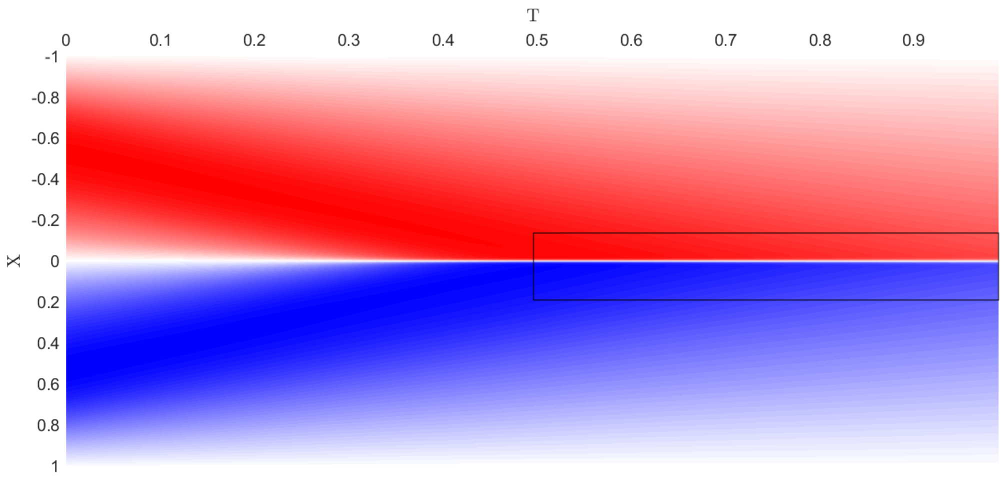


Figure 50: Exact solution with the collocation refinement area remarked

The performance comparison between network 3 and network 4 of the table set above can be observed in figure 51. This performance is evaluated with the convergence behaviour of the two solutions, both in L_2 error and in loss value.

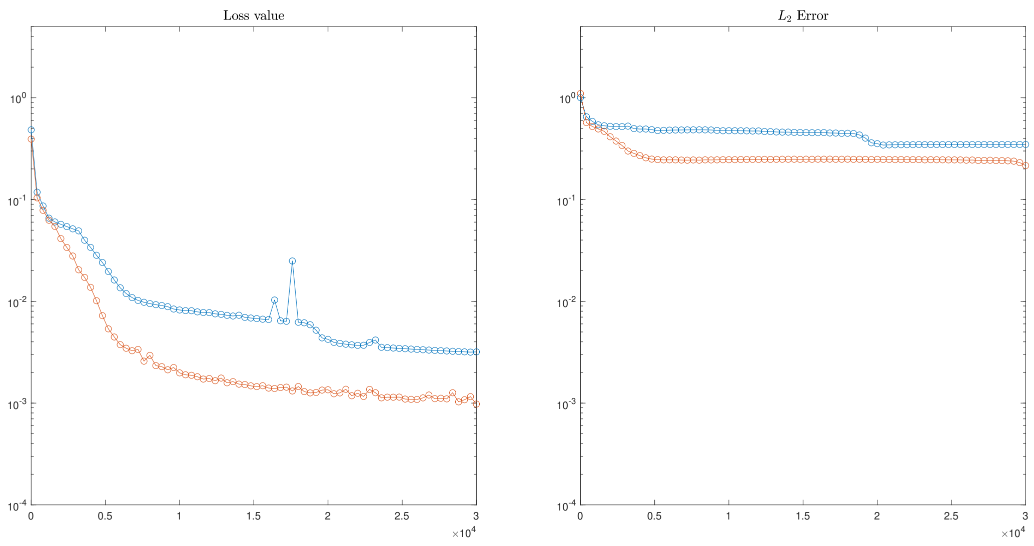


Figure 51: Convergence comparison between the PINN with refinement (blue) and the PINN without refinement (orange)

The error not only does not get better, but worsens with the refinement. Distributing the collocation points in this way did not contribute to get a better prediction in terms of error. Furthermore, looking at the local error distribution in the refined case in figure 52, the observed results worsen.

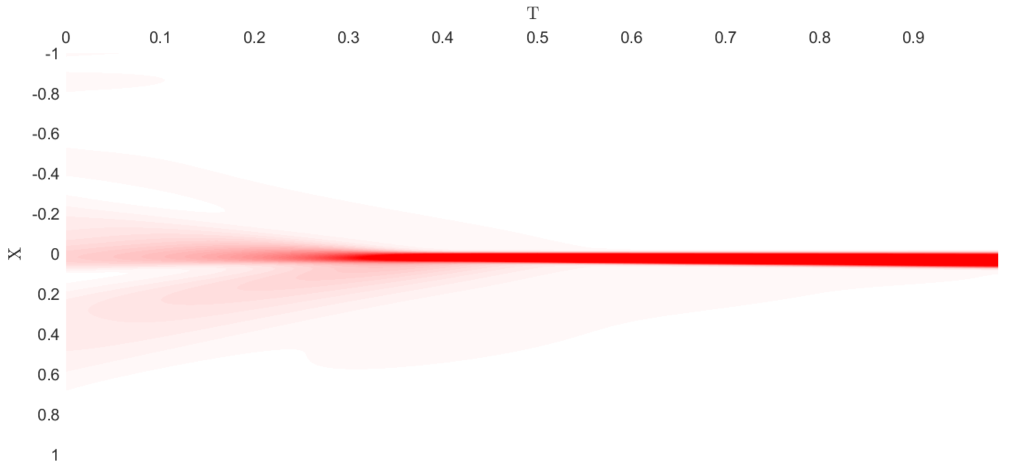


Figure 52: Absolute error distribution of network 3

The main source of the error appears to be the area where more collocation points where introduced. In order to ensure that this error behaviour is due to the location of the refinement. A different refinement is implemented following the structure of figure 654. The only parameter changed is the location of the area where the "extra" 2120 collocation points are introduced. The new area proposed can be observed in figure 53

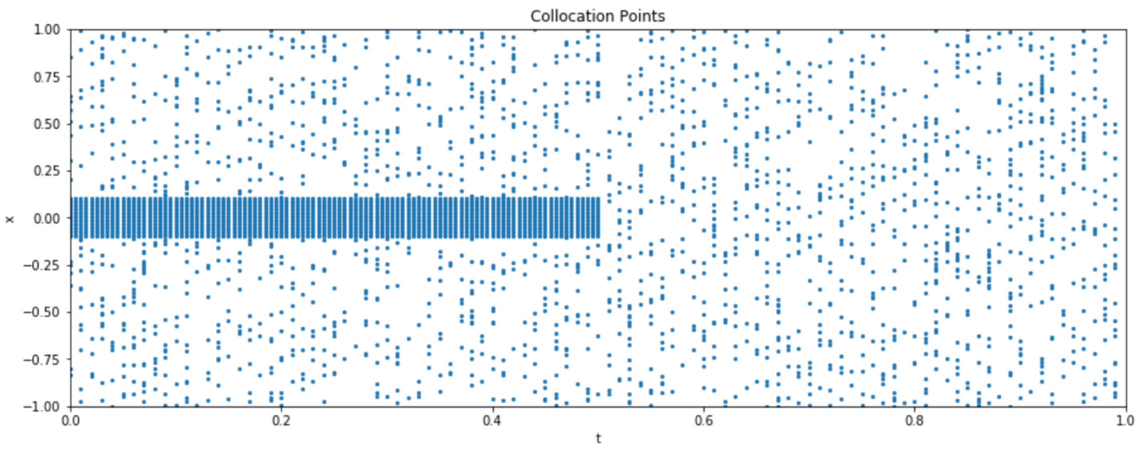


Figure 53: Collocation point distribution with 4000 collocation points

The reason for selecting this collocation point distribution is that, due to the nature of the PDE, the refinement in a zone that is previous in time to the conflict (the discontinuity) will facilitate the prediction of the posterior behaviour in this conflictive zone. The convergence results compared with with the case where there is not a special refinement can be observed in figure 54.

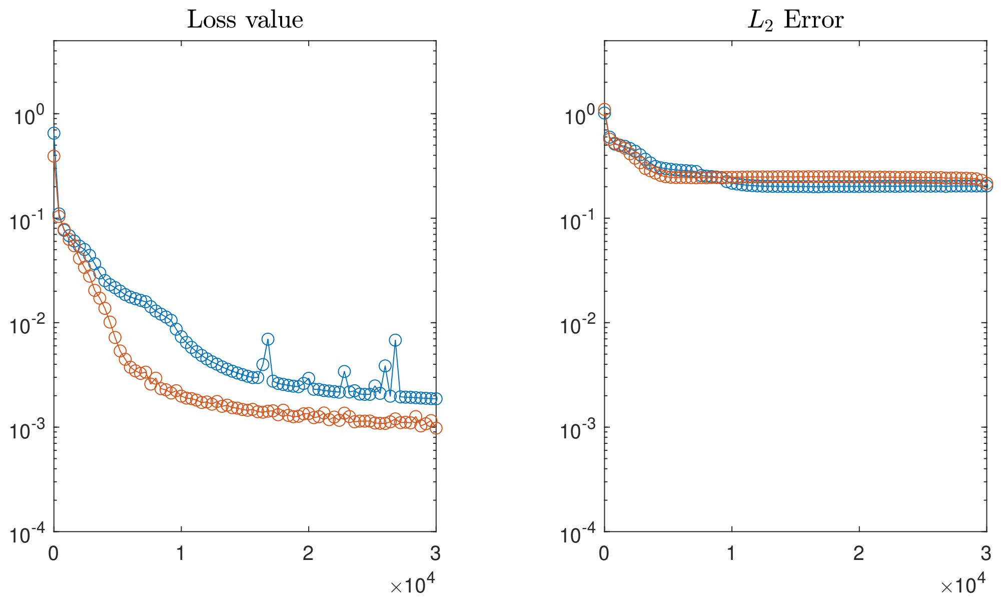


Figure 54: Convergence comparison between the PINN with refinement (blue) and the PINN without refinement (orange)

While it is true that the behaviour observed in this case results in a smaller error with respect to the one in figure 51, the PINN in which we have randomly distributed the points still gives an error performance that is as good as the second refinement.

We will now try the same refinement procedure with the difference that now the total number of collocation point will be 8000. This way, the result obtained in figure 43 is comparable to the one obtained now. The first collocation point proposal is analogous to the one showed in figure 49, with the difference that now the collocation point density naturally has increased. This structure can be seen in figure 55.

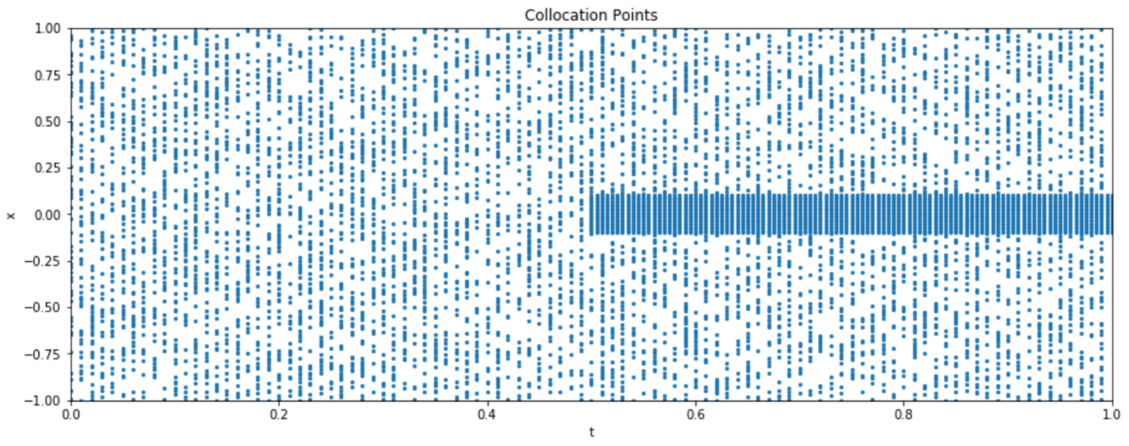


Figure 55: Collocation point distribution with 8000 collocation points

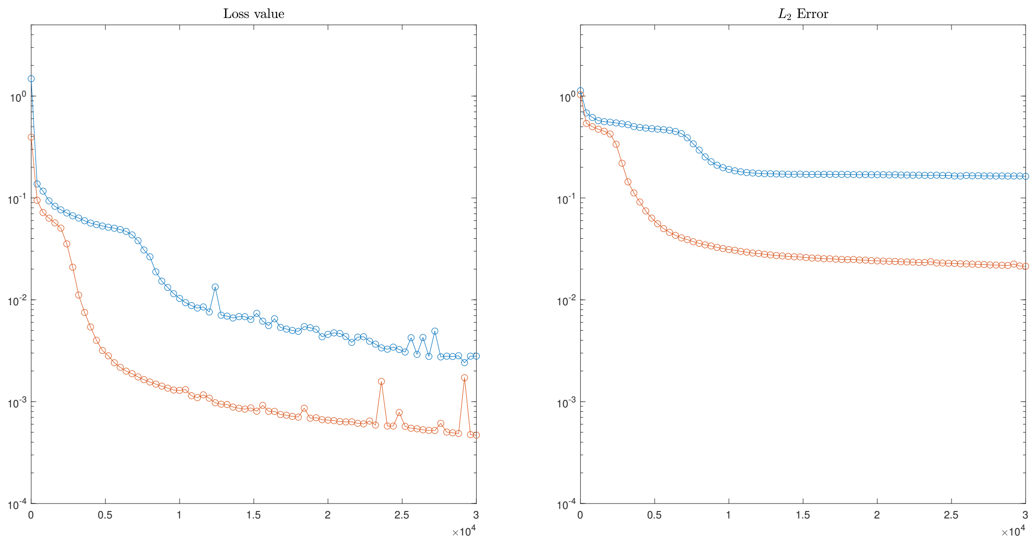


Figure 56: Convergence properties of the refinement with 8000 collocation points

The result obtained is analogous to the one that resulted from the 4000 collocation points case. The local refinement does not contribute to enhance the performance of the algorithm in any case. Moreover, the error that results locally increasing the number of collocation points severely aggravates the error result of the method. Continuing with the procedure executed in the 4000 collocation point case, we now change the location of the refined collocation group with the same criteria.

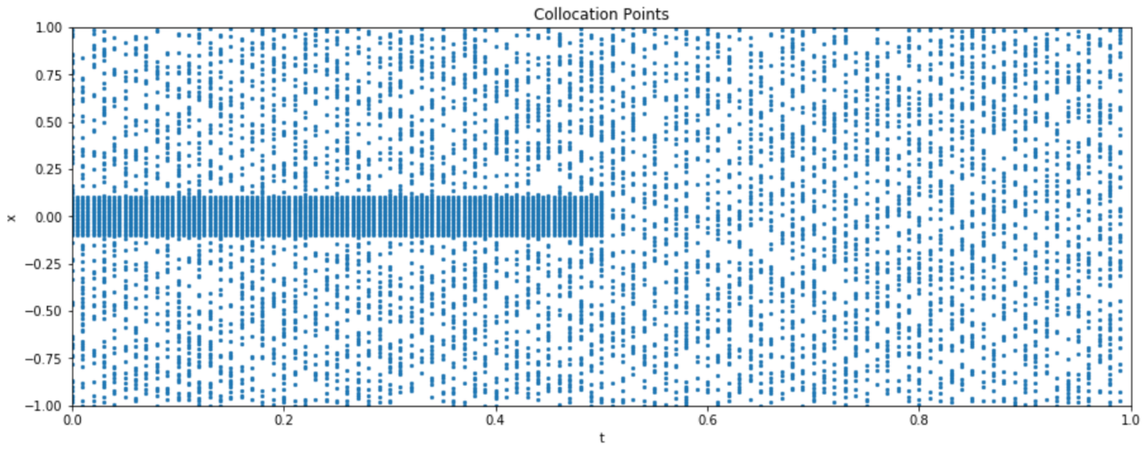


Figure 57: Collocation point distribution with 8000 collocation points

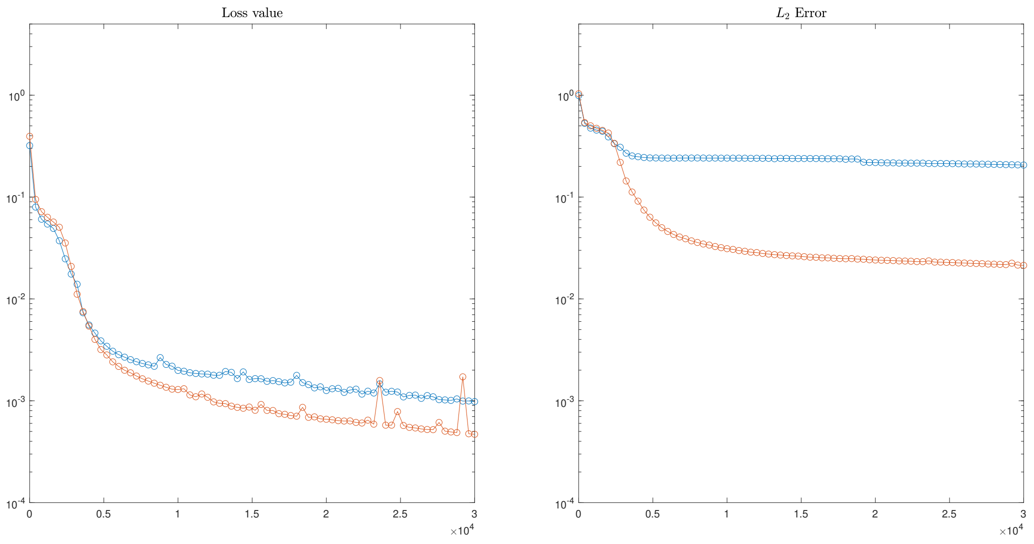


Figure 58: Convergence properties of the refinement with 8000 collocation points

After this example, it becomes clear that, at least for this Burgers example, the local increase of collocation point in the conflictive areas of the problem does not compliment the approximation capacity of the network. It diminishes its possibility of achieving a prediction with a low level of error

13 Singularly perturbed problems

Due to the highly counter intuitive result obtained in the burgers equation, it was decided to pursue a simple study in 1D in order to check whether this collocation point refinement does indeed injure the capabilities of the PINN. To this end, the problem select to which we will apply the PINN is a 1D singularly perturbed problem of the following form:

$$\begin{cases} -\varepsilon u''(x) + u'(x) = 1, & x \in (0,1), \\ u(0) = u(1) = 0, \end{cases}$$

This problem creates a boundary layer that increases its slop for decreasing values of ϵ. This behaviour can be observed in figure 59.

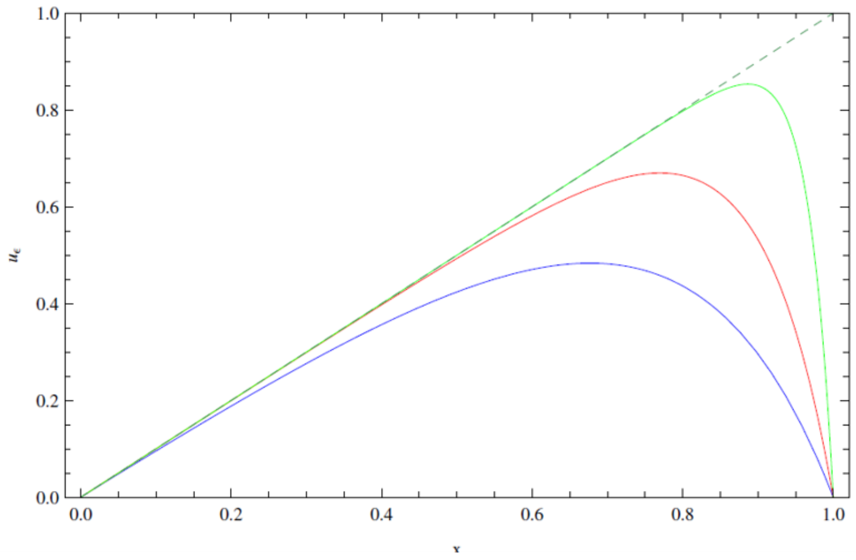


Figure 59

In this figure, the sequence of the graphs: blue, red and green corresponds to progressively decreasing the ϵ value towards 0. The solution creates a boundary layer for small values of ϵ. The analytical solution of this problems as a function of ϵ is:

$$u_\varepsilon(x) = x - \frac{\mathrm{e}^{-(1-x)/\varepsilon} - \mathrm{e}^{-1/\varepsilon}}{1 - \mathrm{e}^{-1/\varepsilon}}, \quad x \in (0,1)$$

The solution $u(x) = x$ is perturbed by the ϵ. The smaller ϵ is, the thiner the boundary layer becomes.This problem is very well suited for this purpose because it has an area were in traditional methods, the mesh used is refined in order to achieve more accurate the results.

The objective then is to compare two different PINN set ups whose sole difference is the distribution of the collocation points. The number of collocation points, the network architecture, the optimization setup and the rest of the parameters that have been studied above will remain constant in these two study cases. The foreseen output of the two networks will only depend on the collocation point distribution. The selected value of ϵ is $\epsilon = 0.01$, which presents a considerably thin boundary layer that will present severe difficulties to the PINN.

For both cases, the parameter set defined can be observed in the following table:

ϵ	Collocation points	Training points	Layers	Neurons/layer
0.01	2000	2	3	10

The collocation points have been randomly distributed in the domain. The results of the optimization process are showed in figure 60.

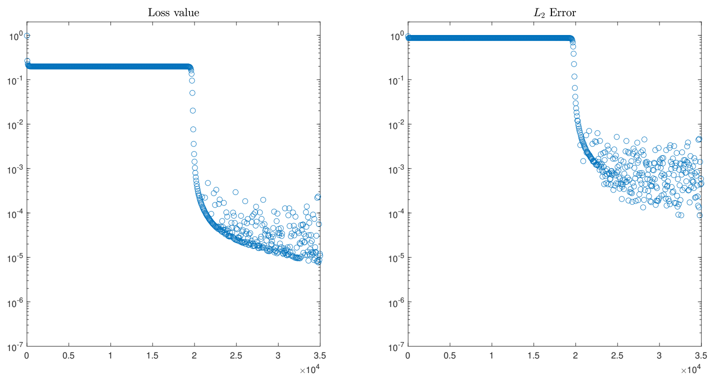


Figure 60: Loss and error evolution of the 1D singularly perturbed problems with no local refinement

The loss and error evolution stagnate at the beginning with high difficulties for the PINN to mimic the behaviour of a singularly perturbed problem. After a certain number of iterations, a "cliff" shape appears. The network quickly converges towards the solution.

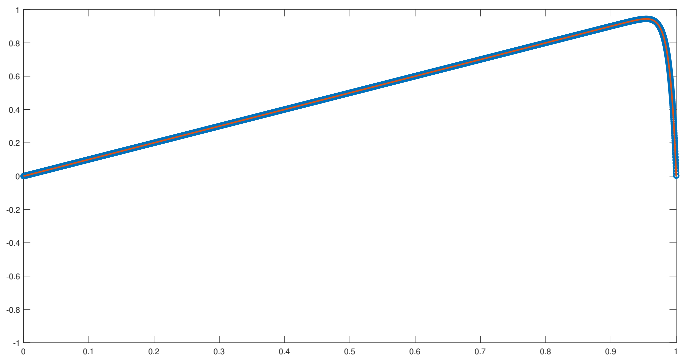


Figure 61: Prediction and exact solution

The idea now is, with the same network parameters, to distribute the collocation points the following way: 1000 collocation points in the length that goes from $1 - 6\epsilon$ till 1, and the rest of the collocation points located on the remaining domain.This way, we form a refinement were the boundary layer is located. This distribution is displayed in figure 62.

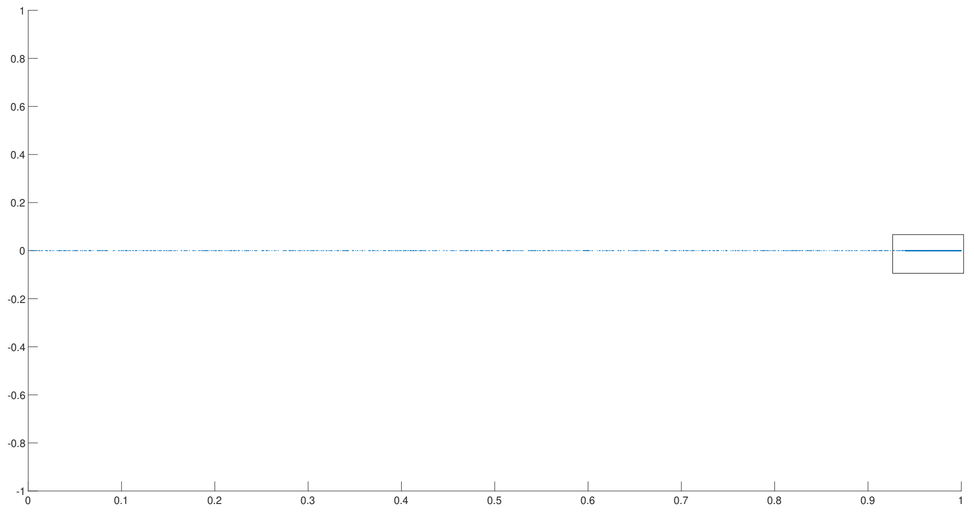


Figure 62: Collocation point distribution

We can observe a severe refinement in the local area where the boundary layer appears in the solution. The optimization process with this collocation point structure yields the following results.

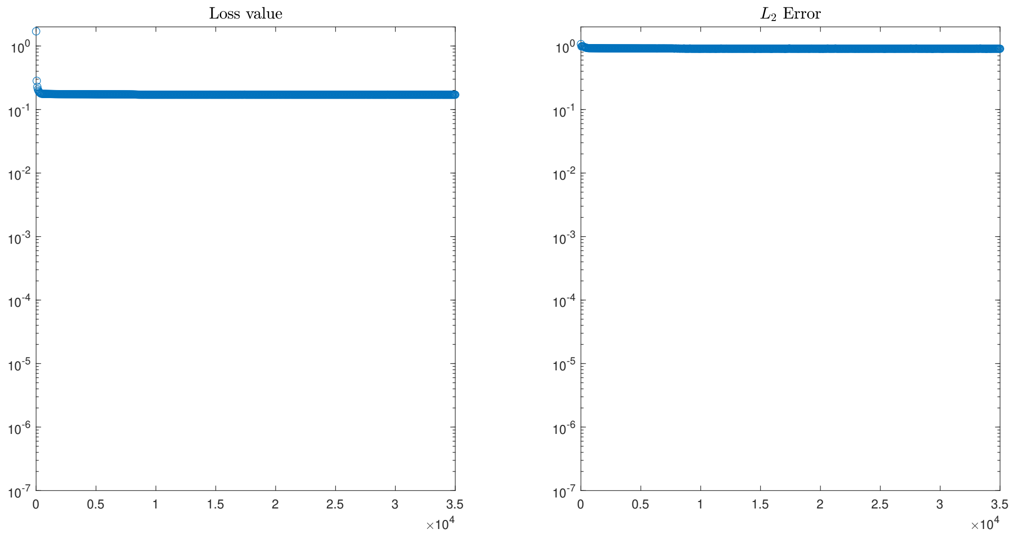


Figure 63: Loss and error evolution of locally refined collocation point PINN

The prediction that the PINN outputs in this stagnation stage can be observed in figure 64. The blue line represents the prediction with respect to the exact solution (orange).

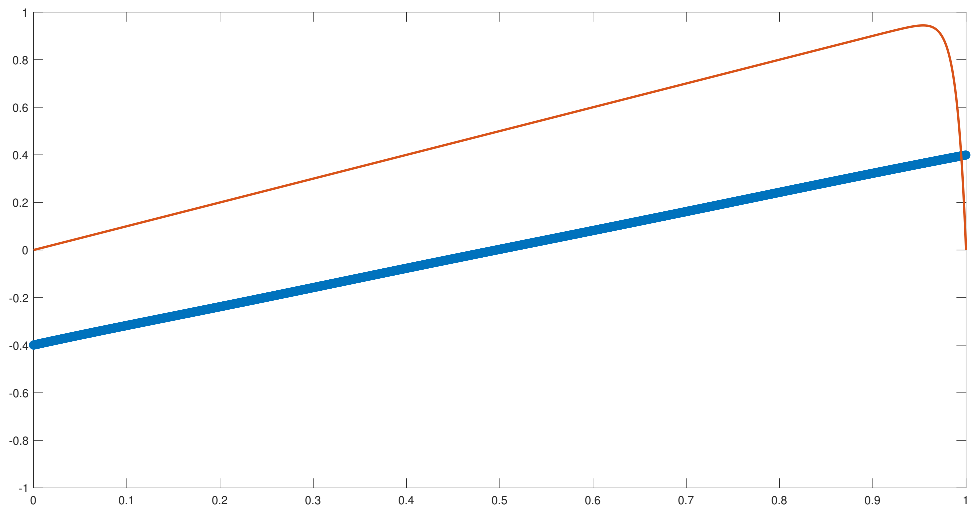


Figure 64: PINN prediction with a collocation point refinement

The PINN fails to converge after 35000 thousand iterations, yielding considerable worse results with respect to the previous study case. The network does not achieve convergence despite dedicating half of the collocation points to enforce the physics of the equation in the boundary layer. The fact that this happens may be because of the fact that each weight that is being optimized in the network affects the value of the network in the whole domain. Optimizing a global analytical expression may make counterproductive optimizations where a local area is enforced. Due to the different aspects that have been covered in this project, it is considered that a more thorough analysis in this fact would be interesting in the future, but it results out of the scope of this project.

One possibility in this matter would be to use activation functions in the network that would only have significant values in local areas of the network, perhaps making it more adaptable to local refinements. In any case, this is just a hypothesis and has not been checked due to the complexity of its implementation and the overall development of this project.

In any case, it has been seen that it is certainly possible that a local refinement in the collocation point distribution of a PINN may very well be counterproductive. This refinement injures the overall approximating capabilities of the network and severely increases its error up to the point where the network does not converge to the desired solution.

13.1 Network Architecture incidence

Before starting the time saving techniques based in the use of a helper loss function that re-utilizes previous results, a final parameter study is going to be investigated. This is the case of the shape of the network. This is, with the same number of trainable parameters, which combination Number of layers - NPL is the one that achieves the best result? Which option takes more optimization time?

With this envisioned question, it has been decided to compare the two following examples.

Network number	Collocation points	Training points	Layers	Neurons/layer	Number of weights
5 (orange)	4000	1000	4	10	371
6 (blue)	4000	1000	2	17	375

Network 5 is the already exposed PINN that achieved results with low levels of error. We can observe again the L_2 error and loss value achieved in figure 47. We will compare the results of this network with an equivalent network with a sufficient number of NPL so that, taking only half the number of layers, the number of trainable parameters stays as close as possible between the two networks. In this case, network 5 has 371 trainable parameters versus network 6 that has 375. Following the color structure of the table above, the convergence evolution of the two examples can be observed in figure 65.

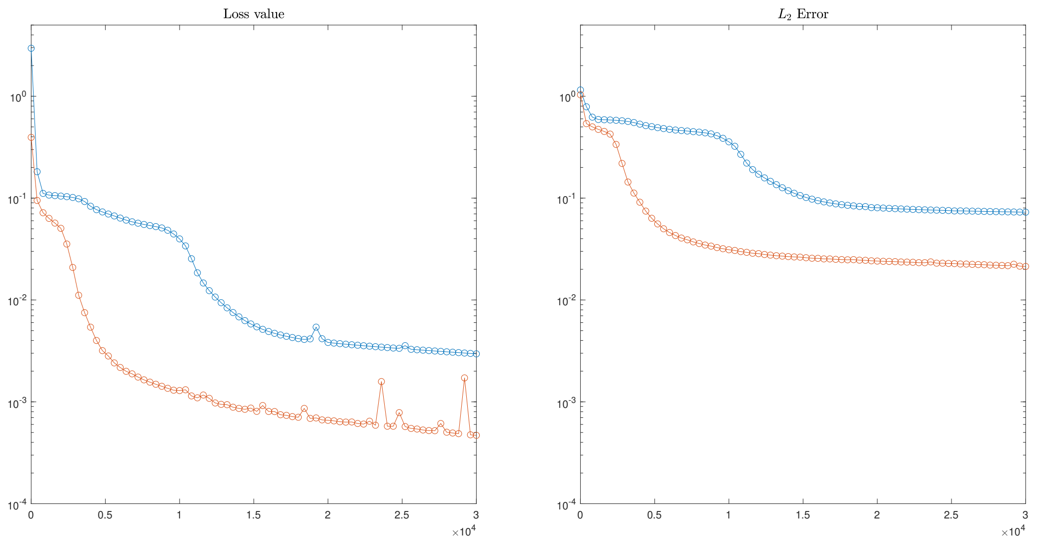


Figure 65: Convergence properties of the two network architectures

The conclusion that can be taken from this example is that with the same number of trainable parameters, increasing the depth of the network is more effective in terms of final error in this case. While it is true that increasing the network width (in NPL) is also effective (there exists a performance enhancement with respect to the 2x10 network) it seems that the complexity of the analytic function created by a deep neural network is better suited for these applications.

The study of the optimization time is also relevant in this matter, we have experienced during this master project that the computation time of the program can be a limiting factor in the computation of the PINN algorithm. In the following table we can observe the output parameters of these two examples.

Network Number	Final loss value	Final L_2 error	Optimization time
5	7×10^{-4}	3×10^{-2}	1931 s
6	3×10^{-3}	7×10^{-2}	2176 s

It is observed that the deeper network, not only performs better in terms of error, but also its optimization time is smaller. Therefore, when the necessity of a more complex network arrises, It seems reasonable to set the way forward first deepening the network, and if that does not achieve the desired results, then increase the NPL parameter.

14 Helper loss function

As we have seen in the many computations done in the previous sections, a limiting factor in the PINN algorithm is its optimization time. The example on which we have based the previous parameter study is a two dimensional scalar PDE. This is, it does not need either a very high number of collocation points nor a very complex network to achieve a good error performance. Still, in some cases the optimization time of the algorithm could take more than an hour.

The objective of this approach is to take advantage of the information obtained by simple networks with a small number of collocation points in order to save time when the intention is to achieve low levels of error.

As in the parameter study, there is a variety of ways we could perform an analysis on the effectiveness of the method, but the objective is clear. Get better measurements of error in a smaller period of time. The pillar of the method, as we have explained in section 345678, consists in taking advantage on the fact that training a network with prior information is much cheaper and faster. Training a network with prior information refers to the training without using automatic differentiation. i.e the helper loss function would look like:

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 \tag{3}$$

Where in this case the solution would be enforced in points inside the domain too. The set of points on which we would enforce the loss function corresponds to the information obtained from the prediction of another network (generally simpler).

The procedure envisioned goes as follows:

1. Choose a network architecture
2. Select the prior information you want to train your network with.
3. Train the network with the envisioned prior information, only trying to mimic the points of your prior information without enforcing any physical law.
4. Select the parameters you want to train your "pre-trained" network with. (Collocation points, boundary points...)
5. Evaluate the error obtained

In this procedure we display the three choices necessary to perform a training with the helper loss function. The first one is the selection of the network architecture, as seen in previous sections, the complexity of this network will determine the minimum level of error achievable. Since the objective of this procedure is to refine the predictions we already have, it seems natural to foresee a higher complexity on the networks selected for this purpose.

Secondly, the next parameter to select it the source of the information to "pre-train" the network. This source of information is mainly characterized by its error. It is important to correctly set these two parameters accordingly so that the network selected is able to refine the solution achieved by the pre-training with the helper loss function.

Finally, the last set of parameters that are subject to selection are the parameters of the networks that have been analized before, specially the number of collocation points. Again, this parameter needs to be aligned with the previous selections, so that the final training leads to a refinement of the final prediction.

It is very important to remark two things about the procedure selected to use the helper loss function. The first one is that an iterative selection of different sets of input information with different sets of network architectures is necessary in order to achieve a thorough conclusion on this matter. This iterative procedure will be analogous with the parameter study made, for each set of input information, an study on different networks with different sets of collocation points will be carried out in order to get meaning-full conclusions.

Before making any computed analysis of the information that will be extracted from each study case, the following hypothesis will be made in order to constraint the selection space on the network architecture of each example: Since the network will start its refinement training with a set of weights that is "oriented" towards the exact solution, it will be assumed that "high bias" errors (due to excessive network complexity) will have more difficulties to appear and therefore, more complex networks are admitted with a smaller number of collocation points with respect to the sets selected in previous examples. It this hypothesis was true, the fact that we have previously oriented the network towards the solution not only would help because there is some optimization path that has already been walked, but also, because the direction of the path has been set, smaller sets of collocation points with bigger networks would be able to achieve higher levels of refinements.

In any case, a set of study cases with different levels of error in pre-trained networks will be selected in order to study these aspects.

14.1 Results

Due to the fact that the results obtained with the use of this tool on different sets of prior information and network architectures did not change, offering similar behaviours, it was decided to present the results in the same section. Consequently, since the results obtained are analogous from one another, we will only present the one described in the table below.

Use of Helper loss function	Collocation points	Training points	Layers	Neurons/layer
yes	8000	1000	8	10

As seen in this table, the input that will be used as prior information in the helper loss function is the case studied in section 12.2. As done before, we now have a two parameter variation capacity, the network complexity (which is considered one variable alone) and the number of collocation point set size (no refinements are considered). These two assumptions have been considered due to the results obtained in the studies of section 12.

The network selected in this case has an added complexity, with a superior number of layers than others considered in previous examples. The intention on performing an optimization with this network is to check whether the hypothesis set in the previous section is correct. This approach is also considered in the number of collocation points, 8000. The convergence results are displayed in figure 66.

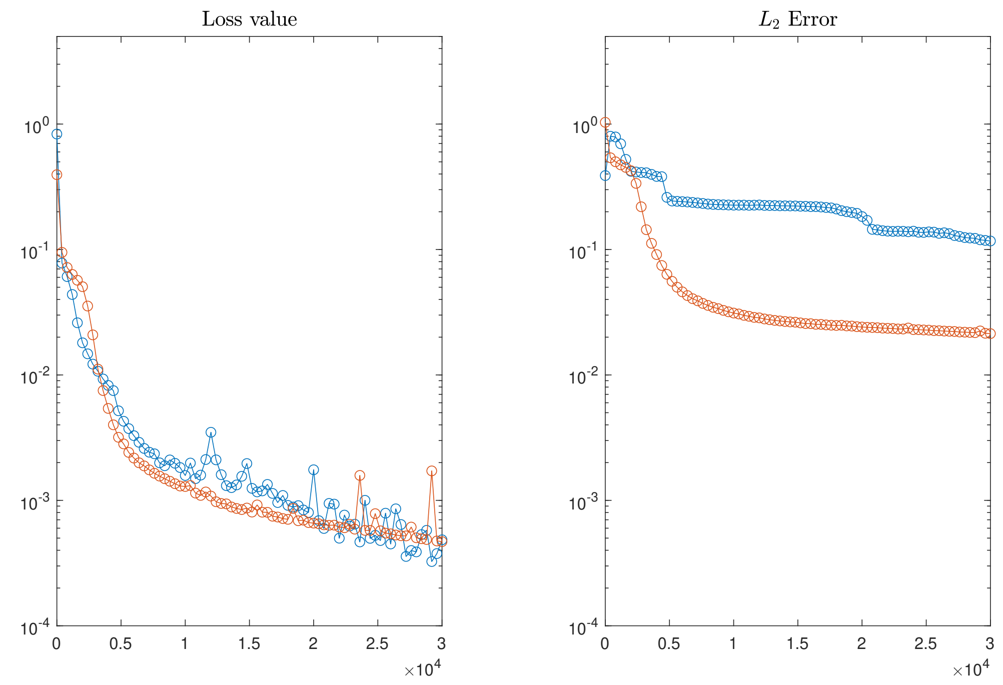


Figure 66: Convergence properties of the two study cases

The conclusion taken from this example (that then was confirmed with different sets of network and collocation points) is that the helper loss function does not contribute to decrease the initial error value of the algorithm. In fact, it does not contribute towards a behaviour enhancement in terms of error.

The results obtained in terms of error have the same characteristics as if the helper loss function was not applied at all. In the case showed, the result obtained in the evolution of the loss function does not translate in terms of error (like seen in previous cases). With a correct parameter tuning, the results achieved are analogous to previous uses.

The explanation of this behaviour can be seen in the evolution of the error in the first iterations. This error evolution in the first stages of the iteration process is displayed in figure 67:

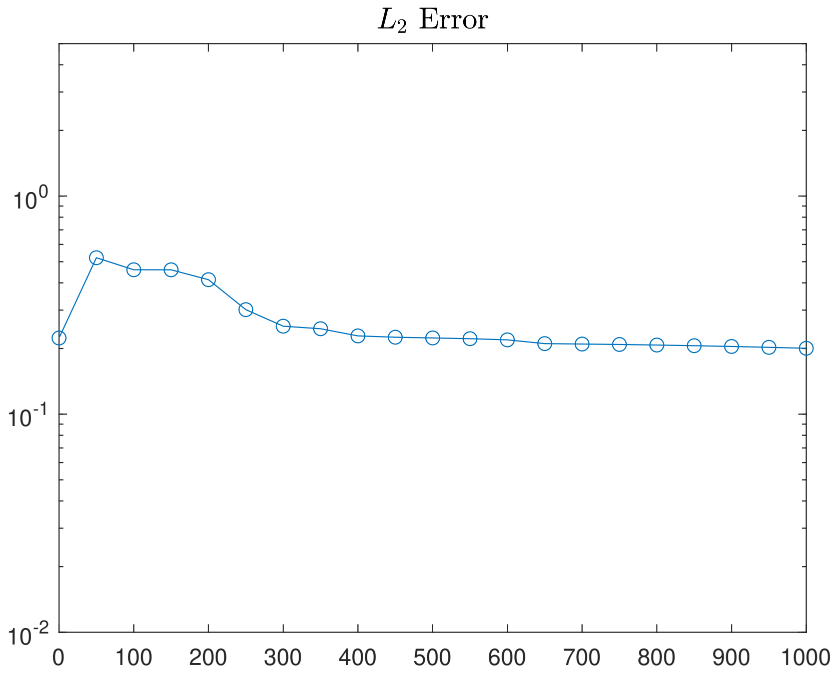


Figure 67: Error evolution in the first 1000 iterations

We observe that, while the initial error without performing any iterations is considerably smaller than what we would usually obtain, during the first iterations the error increases until a maximum value is reached, to then start the proper optimization procedure. The reason of this behaviour is, that the optimization of the helper loss function does not enforce any behaviour in the prediction, i.e. a physical law. Consequently, the initial value of the physical loss once the network has been set with a helper loss function is very high. This optimization of the physics of the problem provokes an initial increase on the global L_2 error, that posteriorly decreases. Enforcing the physical law in the helper loss function too would not solve our problem. The intention of the helper loss function is to save optimization time, which is done with the part of the loss function which does not imply automatic differentiation.

This behaviour does not mean that using a helper loss function does not contribute to the convergence of the problem.We have already checked that when the input domain is big, using the helper loss function to orient the initial weight selection towards the desired solution will help the convergence process. However, this was done to make a first approximation on the initial set on the network. The refinement in terms of error did not arrive with the helper loss function but with the posterior training.

15 Inverse problems

The utility of PINNs is not limited to the data-driven solution of partial differential equations as it has been done before. This machine learning paradigm is also valid to solve inverse problems, i.e data-driven discovery of PDEs [19]. In this study, the attention is shifted to the problem of, given the solution of a certain PDE that depends in a number of parameters, predict which set of parameters best acommodates the given solution with the PDE. Let us consider a nonlinear PDE of the form

$$u_t + \mathcal{N}[u; \lambda] = 0, x \in \Omega, t \in [0, T]$$

being $u(x,t)$ the latent (hidden) solution, and let the nonlinear operator be

$$\mathcal{N}[u; \lambda] = \lambda_1 u u_x - \lambda_2 u_{xx}$$

where $\lambda = (\lambda_1, \lambda_2)$. The question that is formulated in the inverse problem is, given a small set of scattered and potentially noisy observations of the a hidden state $u(x,t)$, with parameters λ best describe the observed data?.

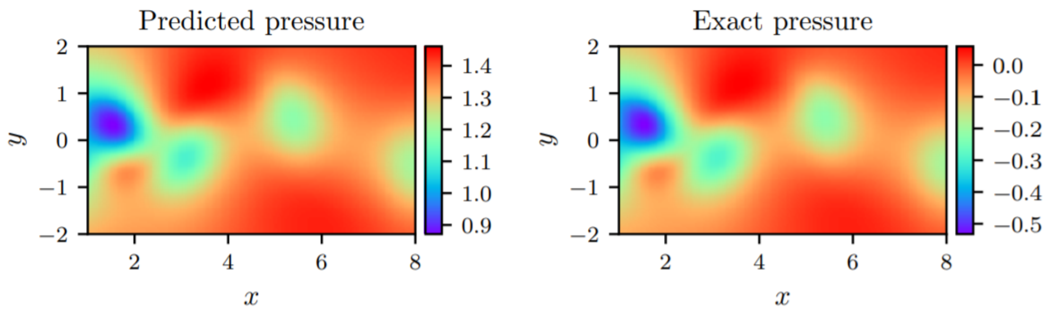
In [19], an overview is provided to tackle the problem. It is assumed that a certain set of measurements across the spacio-temporal domain where the problem exists, is available. The procedure is almost identical to the one used until now. The hidden solution is approximated by a neural network, and the NN is trained to minimize the same two loss functions written earlier in the report

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 \tag{4}$$

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

With two differences.The first one is that now, equation 4 is not only enforced in the boundary, but in the whole domain, being u^i the noisy observations.The second one is that the λ parameters are optimizable variables in the algorithm. Therefore, once the optimization is finished, we obtain the set of parameters whose corresponding PDE best represents the scattered and noisy observations made beforehand. An example where this approach is performed could be the computation of the Reynolds number of a certain fluid flow, given the velocity and pressure field measurements of the problem.

But furthermore, to show what the method is capable of, a example is given in [19] where given scattered and noisy data on the stream-wise and transverse velocity components of a 2D in-compressible NS equations flow, the method is capable of both computing the pressure field and the Reynolds number of the simulation. This result from [19] is shown in figure 68.



Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999(uu_x + vu_y) = -p_x + 0.01047(u_{xx} + u_{yy})$ $v_t + 0.999(uv_x + vv_y) = -p_y + 0.01047(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Figure 68: Inverse problem example from [19]

This result of inferring a quantity of interest from auxiliary measurements by leveraging the underlying physics is a great example of the enhanced capabilities that physics informed neural networks have to offer, and highlights their potential in solving high-dimensional inverse problems.

It is important to keep in mind that, the initial scope of the project is to compute the results of this simulation without any prior data and defining the value of the Reynolds number.This approach has not been pursued in the initial programming effort of the master project, but we have included it in the literature study to give some insight on the versatility of the algorithm we are working with.

15.1 PINN capabilities in Inverse problems

Inverse problems are just another example of the flexibility that PINNs can perform. With the purpose of showing the capabilities of this algorithm, we will perform a study on the parameter estimation of the burgers equation. In [17], the main approach consists in treating the constant value to predict as a variable inside the NN that will be trained with prior knowledge of the solution. This however gives little information about the integrity of the obtained value. It does not specify any statistical knowledge about the prediction, therefore making us unable to evaluate the trust that can be put in it.

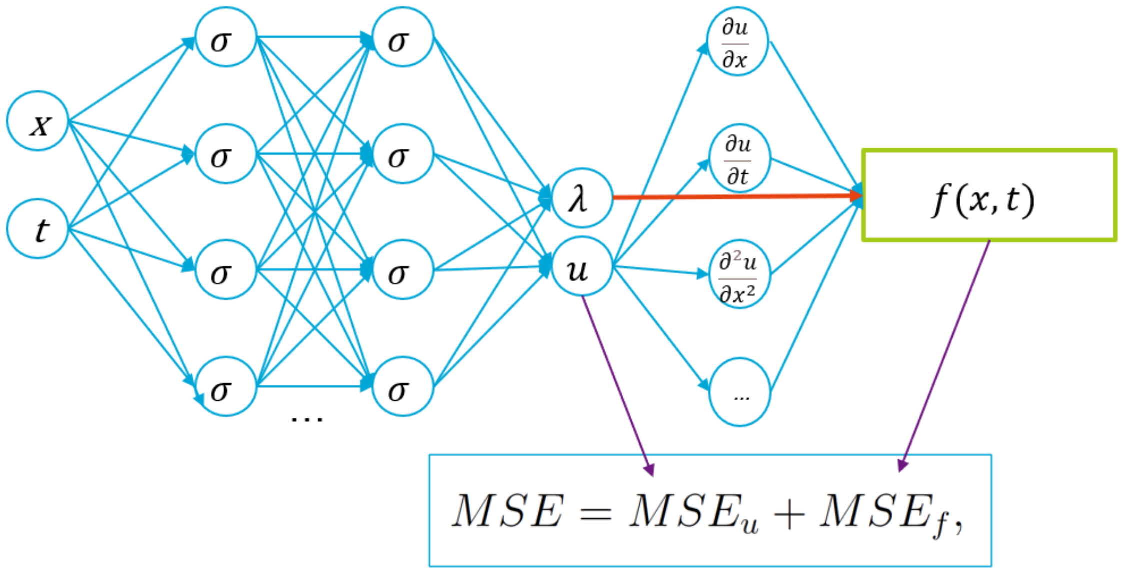
In this project the foreseen approach is as follows. The parameter to estimate is considered a distribution over the domain, i.e, in case we were estimating λ_1 we would assume that this parameter could take different values over different values of x and t.

$$\lambda_1 = \lambda_1(x,t)$$

In order to explain the approach of the PINN in this case, we will apply the algorithm to the burgers example. The parameter to predict is $\lambda_1 = 1$ that multiplies the convective term of the equation.

$$f = u_t + \lambda_1 u u_x - (0.01/\pi) u_{xx},$$

To do this, the foreseen procedure consists in assuming that this parameter λ_1 is a distribution over the domain, i.e. $\lambda_1 = \lambda_1(x,t)$. The neural network then aims to predict two distributions (u and λ_1) instead of one. In direct problems, the distribution to predict was the solution, $u(x,t)$. To do so, we did know the physical law that had to be enforced together with the boundary conditions. In this case, we do not know the complete physical law that needs to be enforced. Instead, prior knowledge of the solution is considered an input in this algorithm. Figure 69 shows an overall scheme of the inverse problem structure.



x
t
σ
...
λ
u
$\frac{\partial u}{\partial x}$
$\frac{\partial u}{\partial t}$
$\frac{\partial^2 u}{\partial x^2}$
...
$f(x,t)$
$MSE = MSE_u + MSE_f,$

Figure 69: Inverse problem architecture

The methods that may involve the prediction of the solution are not subject of this study since it is information already known. The intention of this protocol is to predict the parameter, not the solution. The second distribution to predict by this network is the parameter distribution $\lambda(x,t)$, that is obtained with the enforcement of the PDE in the collocation points. In order not to loose the uniqueness of the problem, the exact solution of this problem is also enforced in this collocation points.

If the exact solution (or an approximate solution with a certain level of noise) was not enforced in the collocation points, the possible solutions of this problem would not be unique. The parameter distribution of the PDE could adapt to a certain solution differing from the intended one at some point. The use of prior information is what guarantees the convergergence of the studied parameter towards the theoretically stablished. In order to clear up the procedure followed, a code extract is shown below.

```
1def f_model(X,model):
2
3
4     lambda_1 = tf.Variable([1.0], dtype=tf.float64)
5     lambda_2 = tf.Variable([0.01/np.pi], dtype=tf.float64)
6
7     with tf.GradientTape(persistent=True) as tape:
8
9         tape.watch(x)
10         tape.watch(t)
11
12         X = tf.stack([x[:,0], t[:,0]], axis=1)
13         out = model(X)
14         u = out[:,0:1]
15         lambda = out[:,1:2]
16         u_x = tape.gradient(u, x)
17         u_xx = tape.gradient(u_x, x)
18         u_t = tape.gradient(u, t)
19
20     del tape
21
22     f = u_t + lambda*u*u_x - lambda_2*u_xx
23
24     return f
```

There are a number of features of this extract that are relevant to explain the procedure carried out. First, it is observed in line 13 that the number of outputs of the model es equal to 2. The model predicts two distributions, u and λ_1, in this code extract named lambda in line 15.

In line 22, the enforcement of the physical law is carried out, it can be observed that instead of the λ value equal to 1 (line 4) we insert the distribution lambda that is predicted by the NN.

```
def loss(u_b, X_b,X,model):

    out_pred = model(X_b)
    u_pred = out_pred[:,0:1]

    f_pred = f_model(X,model)

    a = tf.reduce_mean(tf.square(u_b - u_pred))
    b =  tf.reduce_mean(tf.square(f_pred))

    return  a+b
```

The loss function in this case is analog to direct problems with one main difference, in this case the vector u_b contains prior information of the exact solution inside the domain, represented by the points in X_b.

This set up will allow us to compute the resulting distribution of the parameter λ_1 after a certain number of iterations. If a perfect optimized solution was obtained with a final loss value of 0, the expected solution would be a constant function all over the domain with a value equal to 1. Any deviation from that value is interpreted as an error, either from the lack approximation capacity of the network, an insufficient number of collocation points, or because the loss function obtained is not 0.

It is important to note that in this set up, the convergence problems seen in previous sections severely decrease. The fact that now the training of the analytic prediction $u(x,t)$ is done directly with prior information facilitates the task of both predicting an analytical expression for this function and predicting the parameter distribution. In this latter case, the function to predict has a constant value, relieving the neural network of a boundary layer behaviours difficult to mimic. Again, any behaviour that exits a constant value will be a result of different error sources of the algorithm.

15.2 First Inverse case study

As explained in the previous section, the objective of this first study is to approximate the value $\lambda_1 = 1$ of the convective term in the burgers equation:

$$u_t + 1uu_x - (0.01/\pi)u_{xx} = 0,$$

As explained, we approximate u and λ_1 by a deep neural network, the input characteristics of the PINN are displayed in the following table:

Collocation/prior points	Training points	Layers	Neurons/layer	nº outputs
4000	2000	4	10	2

In this PINN, the prior solution and the physical law are being enforced in 4000 points. In this problem, the characteristic that will be displayed to evaluate the quality of the solution is the evolution of the mean value of λ_1 over the optimization process. Figure 70 shows this evolution:

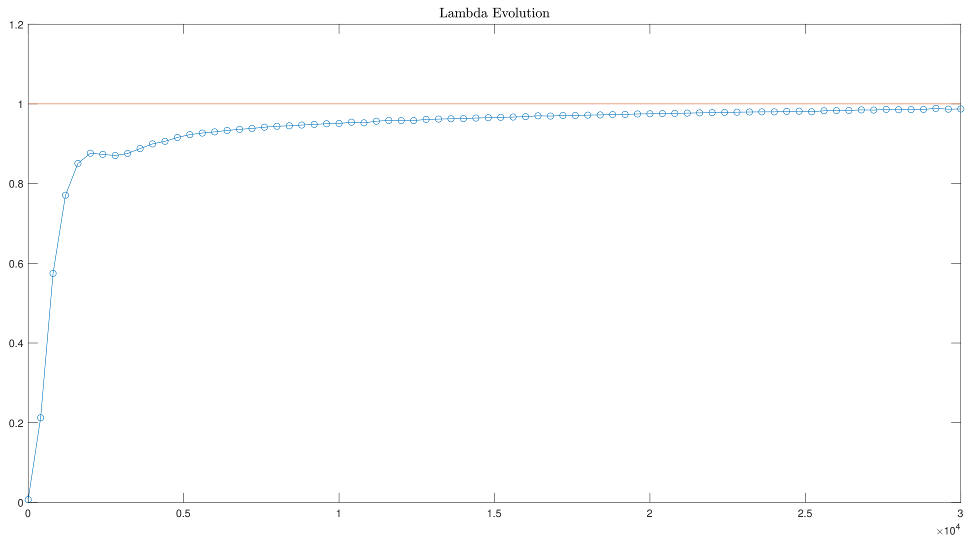


Figure 70: Mean λ_1 value evolution

The mean value quickly convergences towards to $\lambda_1 = 1$ achieving a final mean value of $\mu = 0.989$ and a standard deviation $\sigma = 0.12$. Approximating the probabilistic distribution of the predicted value of the parameter as a normal distribution, the result is displayed in figure 71:

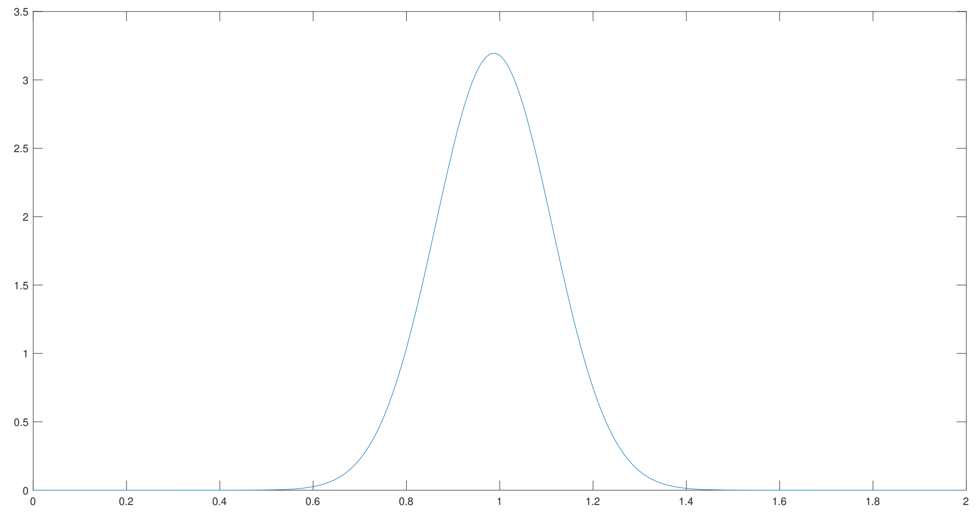


Figure 71: Envisioned λ_1 distribution

This statistical representation is possible to represent becasue a distribution of the parameter over the domain has been obtained. From this parameter distribution shown in figure 72, the standard deviation is computed.

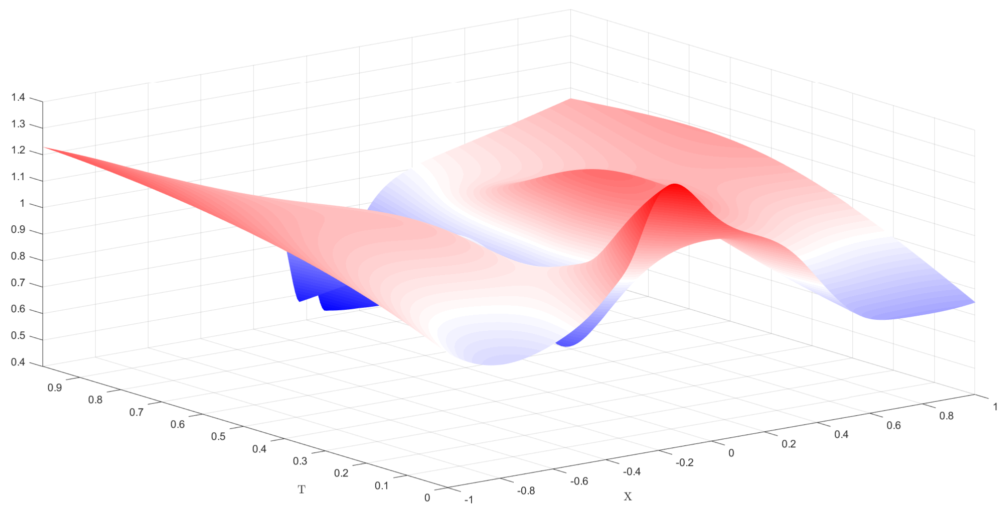


Figure 72: λ_1 distribution over the domain

As explained before, all the deviations from the value $\lambda = 1$ are the result of an error from different sources.in Figure 73, we can observe that the area where the error becomes more significant is were the discontinuity of the burgers equation is found. The colorbar on this graph is set so that 1 represents the color white.Any difference in color is due to the error in the desired value of λ at any given point.

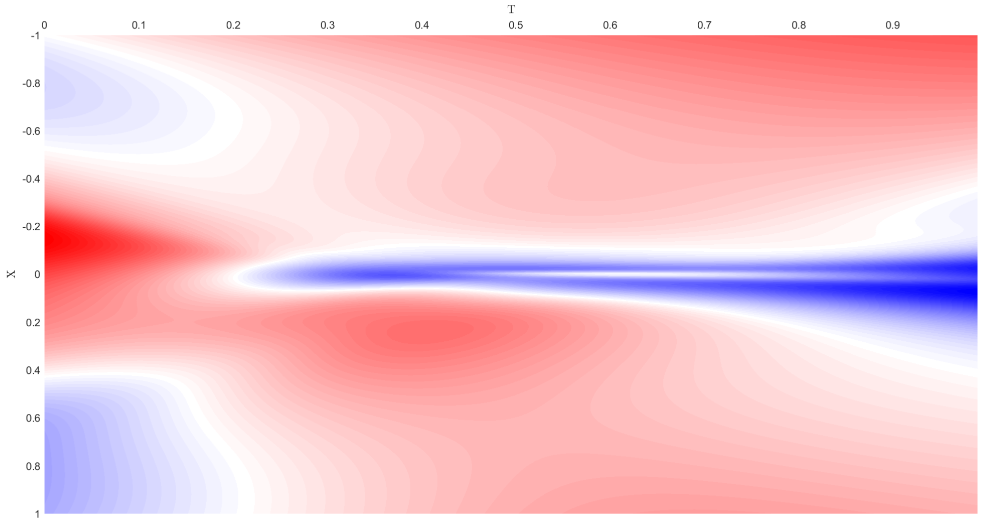


Figure 73: λ_1 distribution over the domain

15.3 Noise generated data

In this example, the exact solution was taken as input to compute the lambda distribution. In practical applications this will not be the case, normally the input generated data has a certain level of noise. In this section the parameter prediction capabilities of PINNs with different levels of noise in the input information is studied.
The noise distribution has been set as a normal distribution with different mean values with a constant standard deviation equal to 1.

The first case study in this matter has a noise with a normal distribution with mean equal to 1. Figure 74 displays this level of noise in the exact solution. The objective is, for the same network parameters defined for the previous case, see how does this affect to the prediction of λ.

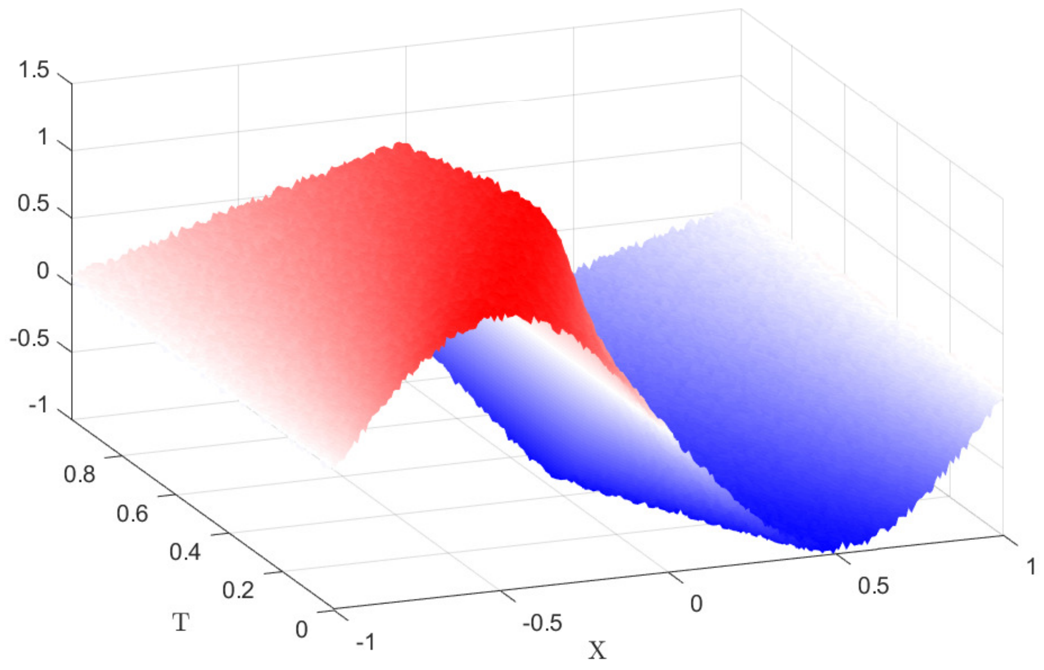


Figure 74: Noise representation $\mu = 1$

The evolution of the mean value of λ_1 with this level of noise is displayed in figure 75. The achieved mean lambda value is $\mu = 0.978$.

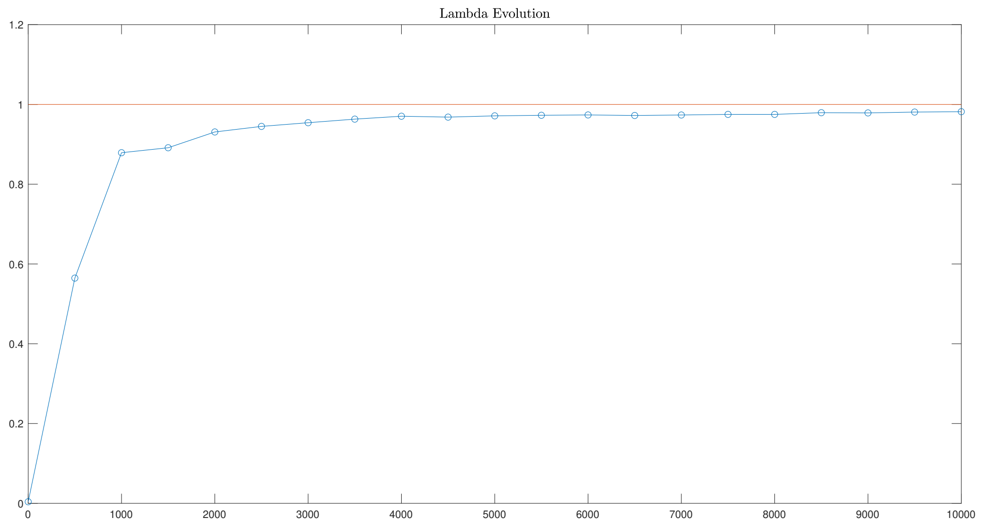


Figure 75: $\lambda = 1$ mean value prediction over the iteration process

The standard deviation obtained in this case was $\sigma = 0.16$, obtaining a slightly wider normal distribution than the case with no noise.

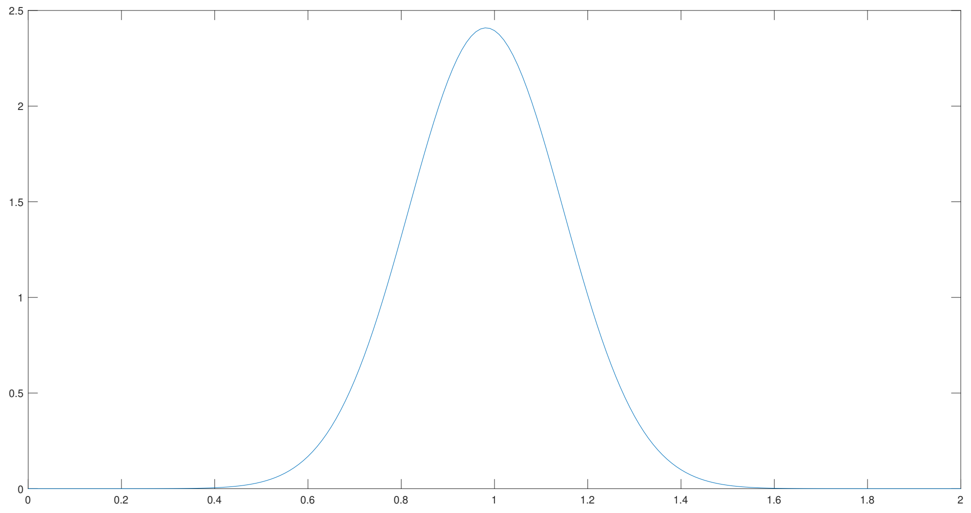


Figure 76: λ normal distribution

It is important to note that despite the noise level defined, the mean value accurately predicts the λ value. The incerased value of the standard deviation can be appreciated in the overall distribution of the parameter inside the domain. This behaviour will be accentuated with increasing levels of noise.

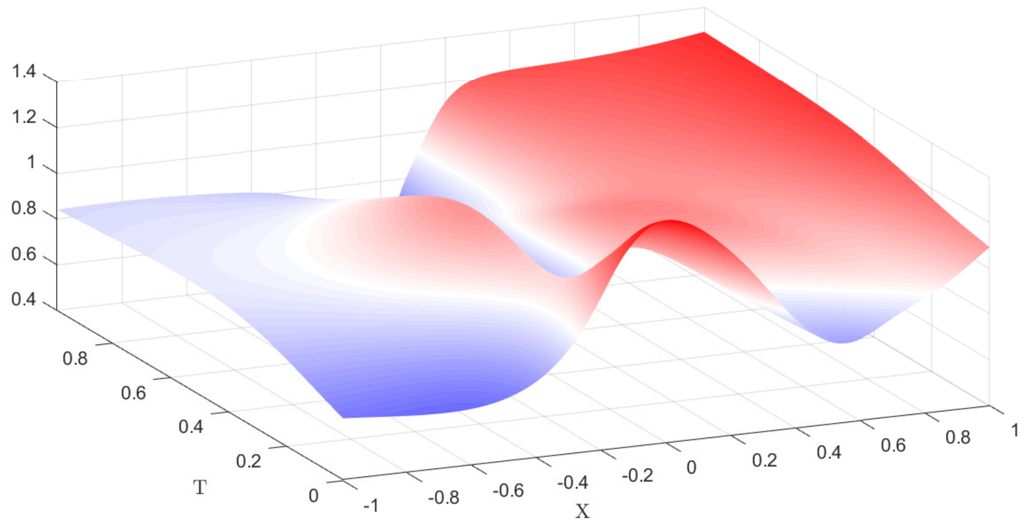


Figure 77: λ distribution in the domain

In order to evaluate the robustness of this method, the noise level is increased with a normal distribution with same standard deviation but higher mean, $\mu = 0.5$. This level of noise is observed in figure 71.

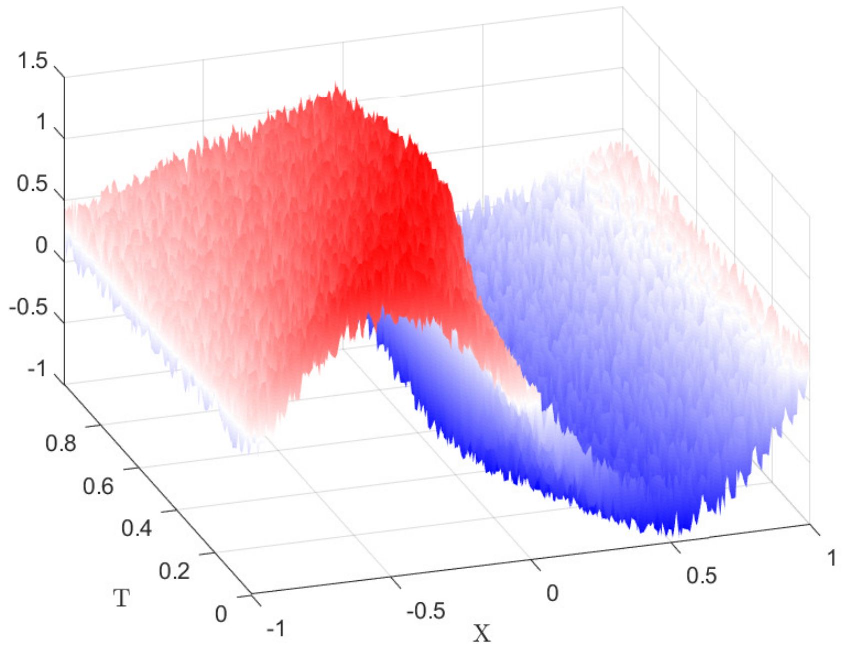


Figure 78: Noise level corresponding to a normal distribution with mean $\mu = 0.5$

The convergence of the mean value of λ_1 over the optimization process is shown in figure 72. The final mean value obtained is $\lambda_1 = 1.06$.

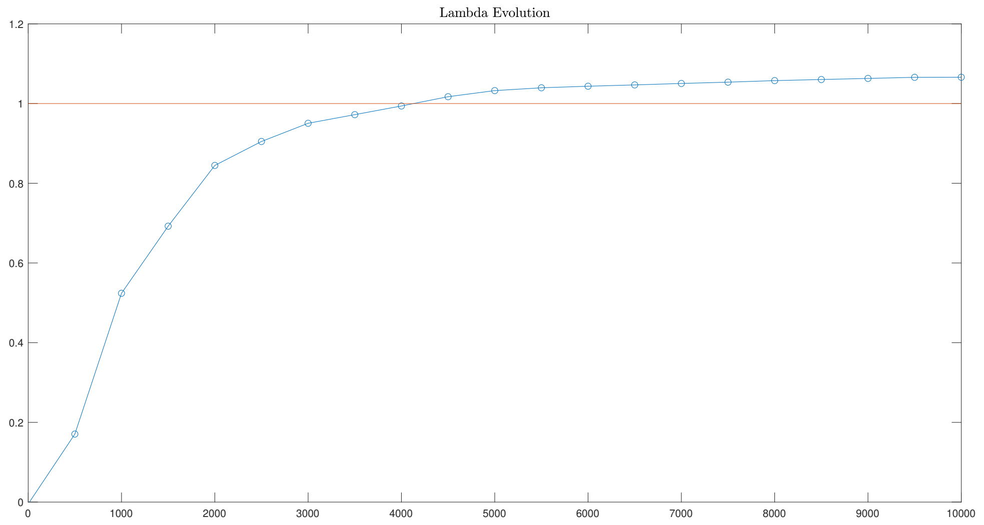


Figure 79: $\lambda = 1$ mean value prediction over the iteration process

The standard deviation obtained is, predictably, bigger. $\sigma = 0.46$. In figure 73 it is shown the distribution of the noise egual to 0.5 with respect to the one previously obtained.

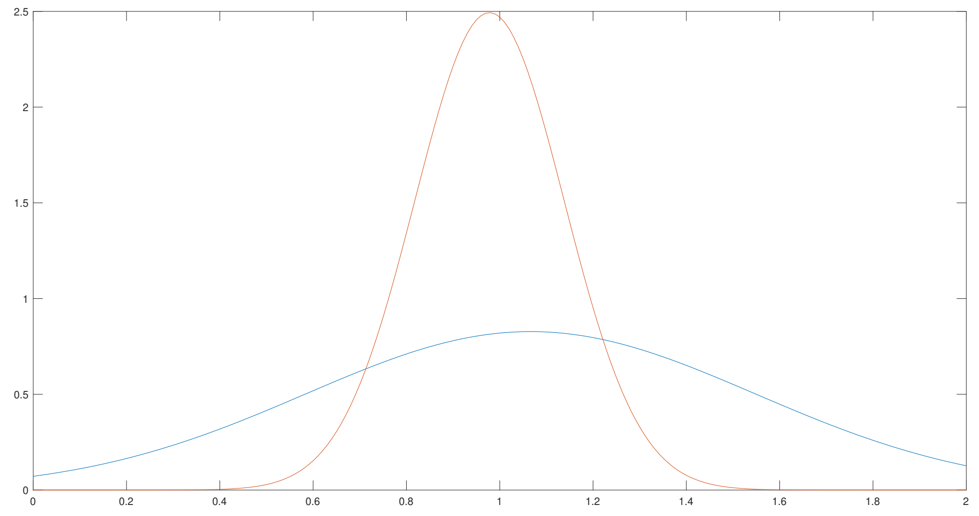


Figure 80: $\lambda = 1$ prediction distribution with respect to the one with noise equal to 0.1

In this case, the predicting capabilities of the network severely decrease. This parameter output can be better explained with the spacial distribution of the parameter over the domain.

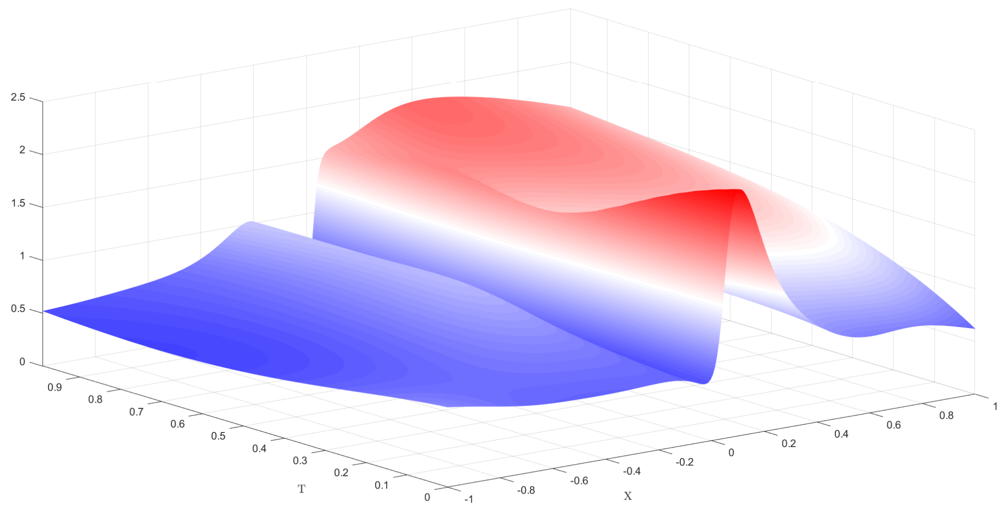


Figure 81: λ distribution in the domain

It is important to remark the high levels of noise that are being introduced to the prior information that the PINN takes into account in order to estimate a parameter distribution. In any case, to continue with this dynamic, the preeidciton capabilities are put to the limit with a noise ratio of 0.8. It is important to take into account that the maximum value of the burgers solution is also equal to 1. we are introducing signal to noise ratios of almost the same order to test the sensibility of the PINN. (we should put somewhere that the same parameters in the network have been used).

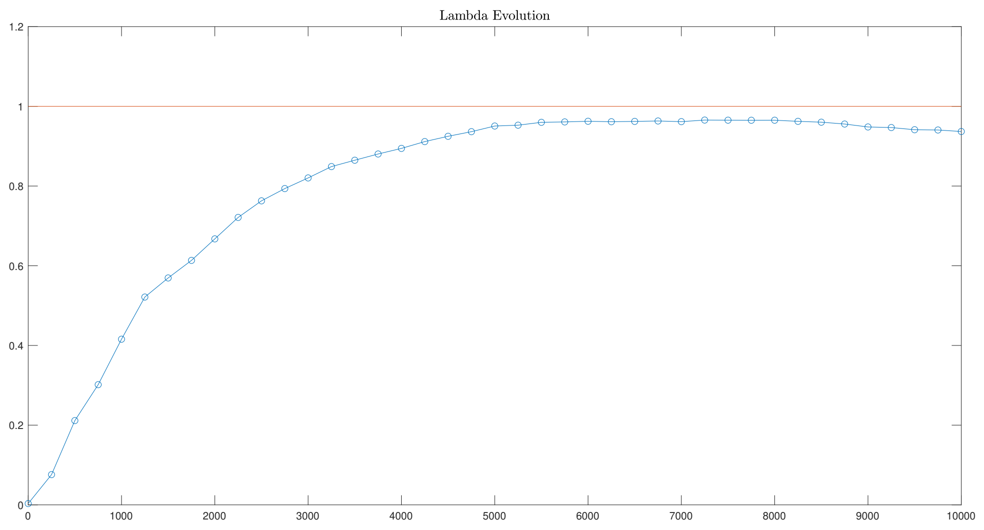


Figure 82: $\lambda = 1$ mean value prediction over the iteration process

The resulting lambda distribution with a normal noise centered in $\mu = 0.8$ becomes (in blue):

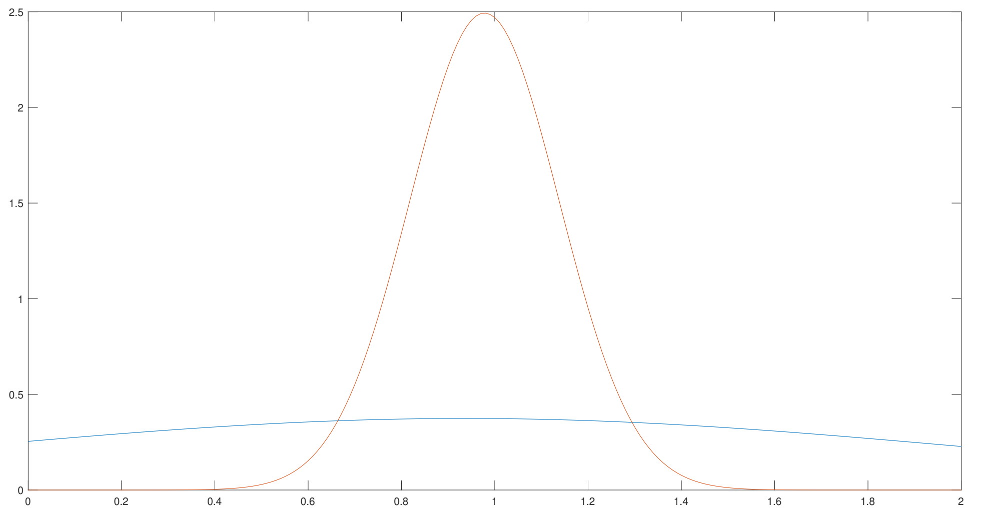


Figure 83: λ distribution

15.4 Conclusions on the noise effect in parameter estimation

The main conclusion taken by observing the distribution characteristics of the parameter estimation over the domain, is that the noise deeply affects the uncertainty on which the prediction is set. The PINN is able to accurately predict mean values of the parameter throughout the domain, but it performs this procedure with increasing levels of standard deviation.

The noise deeply affects the levels of uncertainty obtained for the prediction, decreasing the estimating capacities of the network. It is important to note the considerable levels of noise that the network is capable to overcome.

There are two main observations that are made after this noise study. The first one is that the atention of this noise study has been turned into the accuracy on which the network is able to predict the value of a certain parameter. It is important to note that the network succesfully detects that there is a convective parameter in this partial differential equation. If this parameter was not existent in the equation whose soultion was priorly obtained, this parameter would result a distribution with mean 0. This way, if the differential equation that sets the the physics of a certain solution that is known, a PINN could be used to predict which parts of a proposed PDE play a significant level in the physics of the problem.

There are however two foreseen problems in this approach. The first problem has to do with the possibility that the paramameter to be detected is small enough so that the PINN fails to detect it because of it lack of precision, considering that that parameter is not significant. This could be the case of the diffusive term of the burgers equation. In the previous example of the burgers equation, the value set in the diffusive term is equal to: $\lambda_2 = \frac{0.01}{\pi} \approx 0.0318$. This parameter would enter in the error margin of the PINN capacity as seen with the mean values of λ_1.

The solution that was proposed in this case is to substitute the diffusive parameter λ_2 in the equation by:

$$\lambda_2' = \frac{\lambda_2}{1000}$$

This way, with the same given prior solution, the parameter λ_2' would converge towards the value $\lambda_2' \approx 3$. The prediction of the diffusive term of the burgers equation using this technique is shown in figure 84. The orange horizontal line represents the theoretical value of the equation;

$$\lambda_2 = \frac{0.01}{\pi}$$

The NN of the PINN achieves convergence towards the value of the parameter. The final value achieved after 30000 iterations is $\lambda_2' = 3.74$, i.e. the predicted value with this approach is$\lambda_2 = 0.00374$ with respect to the theoretical value $\lambda_2 = 0.00318$. This method results a success to identify small terms as for instance, convective terms of high Reynolds Numbers.

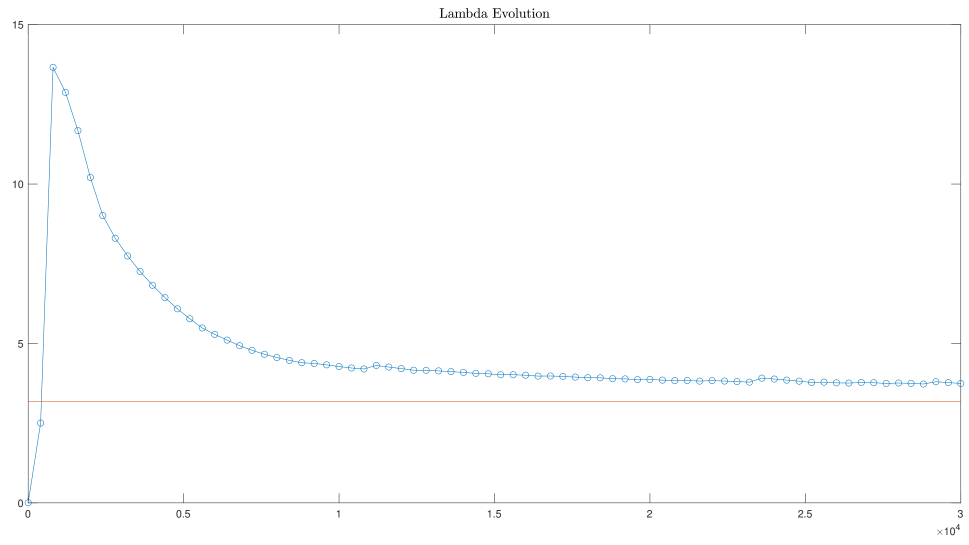


Figure 84: λ_2 mean value evolution

The values of the predicted diffusion lambda over the parameter are:

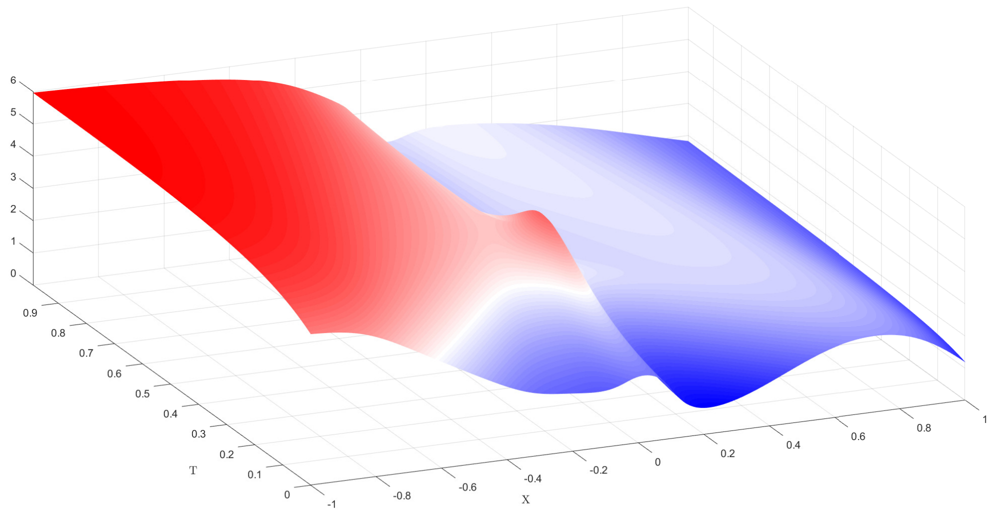


Figure 85: λ_2 domain distribution

And its normal statistical distribution is displayed in figure 86:

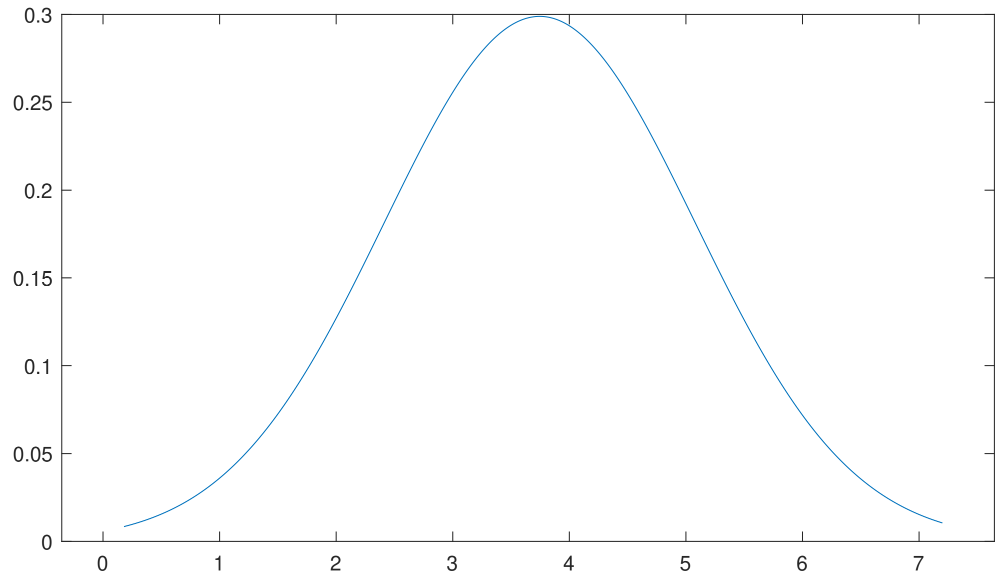


Figure 86: λ_2 normal distribution

While it may seem that this distribution results wide and does not accurately predict the diffusion parameter, it is important to note that the probability predicted that this parameter is equal to 0 in the PDE is negligible. This prediction guarantees that there is a diffusion term in the solution explained.

Continuing with the same idea of identifying the relevant parts of a certain PDE, a problem that we main encounter is the following; the parameter that we are trying to estimate is not a scalar in the PDE by a nonlinear distribution. This distribution could have mean equal to 0 and therefore the conclusion could be that this parameter was not significant in the equation despite it clearly is. This is the case of the following approach: We do not know if there is a term that depends in u_x in the solution of the Burgers equation (We obviously know that it exists, this case is set to see the PINN capabilities),to explore wheter this parameter is important in the equation, we set the following equation to enforce in the PINN:

$$u_t + \lambda u_x - (0.01/\pi)u_{xx} = 0,$$

If the solution previously known of the Burgers equation was applied to predict the parameter lambda only looking at its mean value, the obtained solution would be 0, the mean value of the solution u is equal to 0. However, because what we obtain from the inverse problem is the distribution of λ throughout the whole domain, the obtained information has the distribution showed in figure 87.

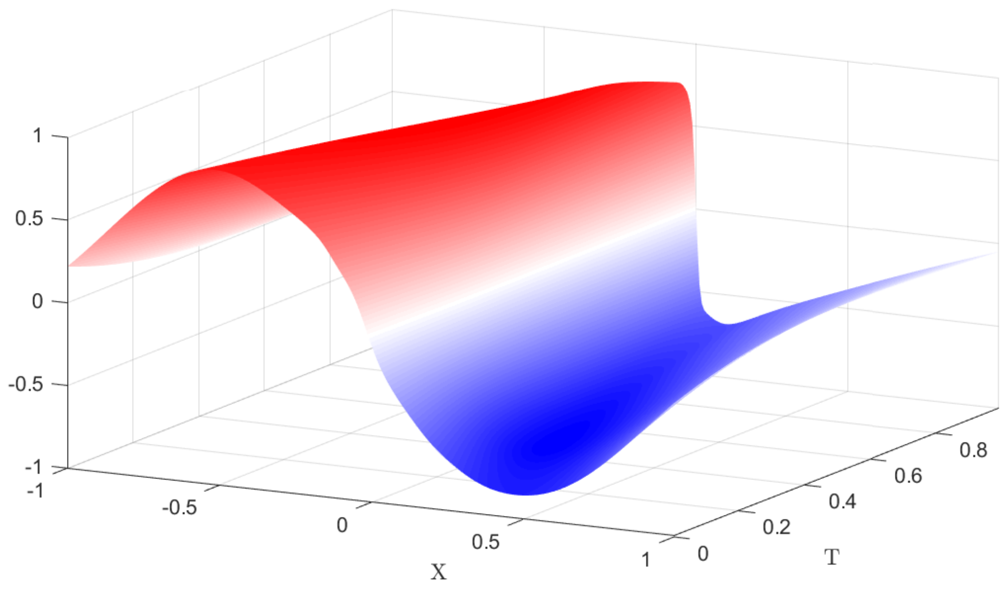


Figure 87: λ_2 domain distribution

The distribution corresponds to u, the nonlinear parameter of the Burgers equation, therefore predicting the equation:

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0,$$

The PINN is able to directly predict the nonlinear terms of the equation because we predict the λ value in each point of the domain. This fact makes PINNS a useful tool not only to predict unknown terms of a certain partial differential equation, but also allow to identify non-linearities in these parameters in case a certain part of a PDE is not multiplied by a scalar value. The parameter distribution prediction is clear, the non-linearity is perfectly shown.

The important fact about this capacity of inverse problems applied in PINNS is that any possible term of a differential equation can be predicted. If we did not know that there existed a convective term in the burgers equation, we could substitute it in the inverse problem by a parameter λ this way:

$$u_t + \lambda - (0.01/\pi)u_{xx} = 0,$$

Figure 88 shows the resulting distribution of trying to predict the convective term. The fact that what we predict is a distribution, allows the PINN even to check whether there is any additional term present in the physics of the problem, and if there is, express its shape. It is important to express that there needs some part of the equation know, otherwise, the resulting equation would be

$$\lambda = 0,$$

And we would be only minimizing the distribution's value in all the collocation points.

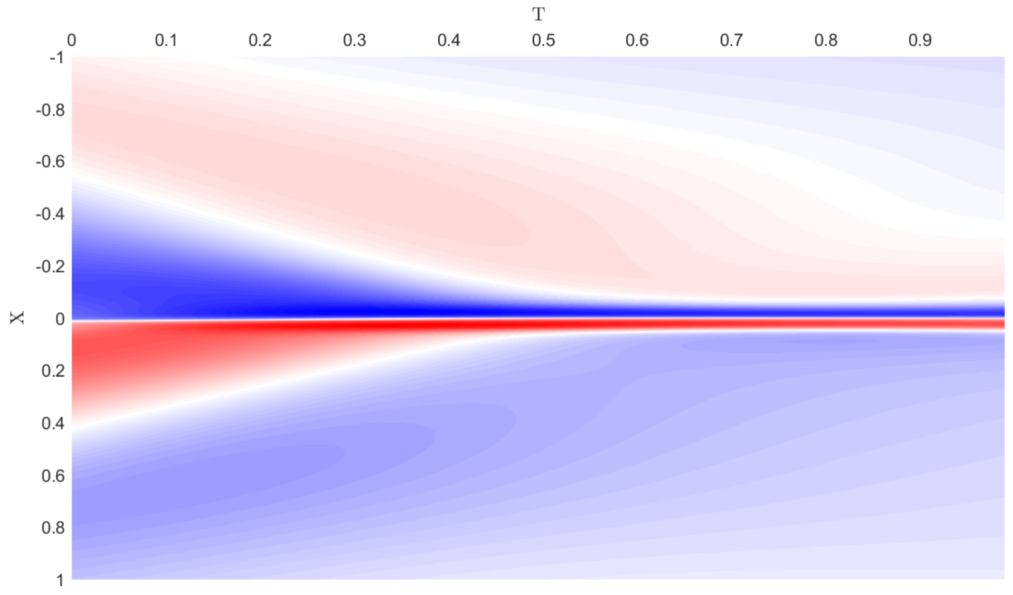


Figure 88: Convective term prediction

As a conclusion on the versatility of PINNS in the use of inverse problems, it is important to note that the fact that not only a scalar value, but a distribution make this tool highly useful for PDE identification. Not only for parameter identification as done in [17] but also to identify possible non-linearities unaccounted for.

This tool can operate with obtained solutions that have high levels of noise, as explained in section 14.3. The way the problem was set, being λ a predicted distribution in a given set of points, it allowed us to compute statistical parameters and construct a probability density function to predict the possible values of the real parameter and its probability. The fact that our parameter is set as a distribution also allows us to evaluate the certainty (i.e. standard deviation) of the mean value obtained.

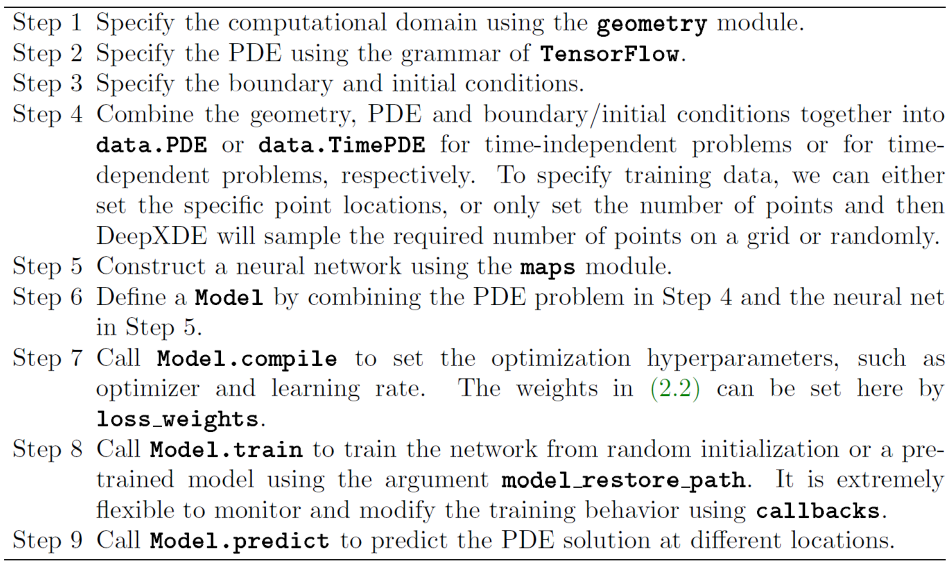
This approach can also be useful in case an experimental solution of a fluid flow equation. In this case, it may be important to obtain which parameter does play an important role on the generation of the obtained solution. This way then, the experimental behaviour of a certain solution can be evaluated and reproduced with the parameters that regulate the solution, simplifying the overall process, deleting the non-significant terms of the equation.

16 DeepXDE

Before proceeding to the conclusions that can be drawn from the initials observations concerning PINNs in this master project, it is interesting to show the efforts that have been done towards developing a PINN simulation library. This library has been developed by [11]. The main reason for developing such a tool is, that the PINN algorithm is simple and it can be applied to different types of PDEs. The DeepXDE tool is designed to serve both as an education tool to be used in the classroom as well as a research tool for solving problems in computational science. Specifically, DeepXDE is able to solve forward problems given initial and boundary conditions, the same objectives we have initially performed in this literature study.

The developers of DeepXDE state that the tool supports complex geometry domains based on the technique of constructive solid geometry, and enables the user code to be compact, resembling closely the mathematical formulation. In sum, this library contributes to the faster development of the emerging Scientific machine learning field.

Compared to traditional numerical methods, the code written in DeepXDE is much shorter and more comprehensive, trying to resemble the mathematical formulation of the problem. The library has a number of built in modules which include computational domains (space and time), constraints, data training or neural network architecture. The main procedure for solving PDEs is shown in figure 89, that is directly taken from [11].



Step 1	Specify the computational domain using the **`geometry`** module.
Step 2	Specify the PDE using the grammar of **`TensorFlow`**.
Step 3	Specify the boundary and initial conditions.
Step 4	Combine the geometry, PDE and boundary/initial conditions together into **`data.PDE`** or **`data.TimePDE`** for time-independent problems or for time-dependent problems, respectively. To specify training data, we can either set the specific point locations, or only set the number of points and then DeepXDE will sample the required number of points on a grid or randomly.
Step 5	Construct a neural network using the **`maps`** module.
Step 6	Define a **`Model`** by combining the PDE problem in Step 4 and the neural net in Step 5.
Step 7	Call **`Model.compile`** to set the optimization hyperparameters, such as optimizer and learning rate. The weights in (2.2) can be set here by **`loss_weights`**.
Step 8	Call **`Model.train`** to train the network from random initialization or a pre-trained model using the argument **`model_restore_path`**. It is extremely flexible to monitor and modify the training behavior using **`callbacks`**.
Step 9	Call **`Model.predict`** to predict the PDE solution at different locations.

Figure 89: DeepXDE PDE solving procedure

DeepXDE supports four standard boundary conditions, including Dirichlet, Neumann, Robin , and periodic, and a more general BC can be used using the module OperatorBC. DeepXDE, permits the possibility to use other NN architecures other than the one used during this literature review (Feed forward neural network). The platform is also able to choose different training parameters such as optimizers, learning rate schedules, initializations and regularizations.

It allows for a wide flexibility in the problems and methods of solution that the user may need. The tool is highly configurable and it conforms the whole design process, from geometry definition to the simulation results. This geometry definition is possible because PINNs employ automatic differentiation to handle differential operators, and thus they are mesh-free.

The development of DeepXDE benefits both the education and the computational science communities, as it provides, from start to finish a whole mathematical implementation of PINNs. By introducing the usage of DeepXDE, it is showed in [11] a fully customizable tool to solve PDE problems with high flexibility.

17 Conclusions

17.1 Discussion

The main objective of this study has been provide a valid framework on the behaviour of PINNs. During the literature study, a first approximation was performed, but it was in the master project were a thorough study was made.

In this master project the whole versatility of PINNs has beenn studied. Concerning direct problems (data driven solution of differential equations),a complete parameter study on PINNs was done in order to establish which of these parameters could be useful in order to optimize the behaviour of the algorithm. The way that this was done was by individually varying these parameters in several studies (Section 12) starting for the simplest possible form of each of them. This way, the effect of different parameters was evaluated guaranteeing that the different measures of error in each study case was solely dependent on the parameter that was being studied.

With this approach, it was concluded in the different studies that the networks of high complexity were being overly used in many cases in literature, considerably slowing down the optimization process. This excess of complexity in the networks used in direct problems could mean a high bias error that may result in considerable convergence difficulties. As seen during this project, PINNS are an algorithm very susceptible to convergence problems that make achieving a correct solution non-trivial, specially in problems of high input dimensionality . In this spaces, the number of collocation points to "fill" the space and constraint the solutions of the universal approximator of the PINN exponentially increase, again, leading to possible "high bias" errors. The main objective of the project was to make a necessary exploration so that after reading it, one has a certain idea on how to approach a direct problem in PDEs with a PINN.

The possibility of applying the same algorithm in principle to solve inverse problems (Data driven discovery of PDEs [15]) shows the versatility of the algorithm. In this matter, a noise-based study was carried out to evaluate the robustness of the method in this respect. We could conclude that noise affects the certainty (standard deviation) of the measured distribution. It was possible to obtain such a distribution because the parameter to predict was treated as such, evaluated in every point of the domain. Because we took this approach, it was also possible to use PINNs to detect non-linearities and complete missing terms on a given PDE.

The intention of the research questions given in the literature review was to give a valid framework so that any person that wanted to sart a certain PDE study using PINNs, was able to use this project as a guideline on ”where to start”. The capacity of fairly simple networks to approximate solutions that accurately mimic the exact behaviour considerably simplifies the algorithm and its possible convergence problems with high bias errors.

The utility scope of many tools (e.g. the helper loss function) was also studied in order to check whether these tools did really were translated into a speed-up of the optimization process. This meant a discarding process on the many possible tools that can be used in this algorithm was carried out,to conclude that the pillar parameters on which the initial algorithm is sustained (network structure and collocation points) can achieve the best influence on the final results. The study then focused on the optimization on the magnitudes that better approximated the desired solutions.

PINNs are a rapidly evolving field. The recent advances in GPU processing and the youth of this approach make PINNs a tool with a huge potential over the next years. Let us just remember some of it’s advantages;the method searches for an analytic expression that could be operated with. The implementation of boundary conditions result trivial even for periodic boundary conditions. It can be used for the analysis of inverse problems and the prediction of unaccounted terms on an equation.

Furthermore, the fact that the solution looked for is an analytic expression, makes expendable the construction of a mesh and the correspondent refinements. The development of sufficiently fine meshes for classical methods requires a huge computational effort. Let us not forget that these methods have been evolving for over 50 years. The only input that a physics informed neural Network needs is a set of points in the selected domain where to enforce the partial differential equation through the loss function. The simplicity of the method makes it deeply prepared to make breakthroughs in the field of PDE simulations.

It is important to remark the capacity of PINNS of incorporating known prior information of the hidden solution of the problem to compute other aspects of it. All these aspects make PINNs a highly versatile and powerful tool applicable to many projects concerning scientific machine learning.

References

[1] Atilim Gunes Baydin et al. *Automatic differentiation in machine learning: a survey.* 2018. arXiv: 1502.05767 [cs.SC].

[2] Avrim L. Blum and Ronald L. Rivest. "Training a 3-node neural network is NP-complete". In: *Neural Networks* 5.1 (1992), pp. 117–127. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80010-3. URL: https://www.sciencedirect.com/science/article/pii/S0893608005800103.

[3] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214. eprint: https://doi.org/10.1146/annurev-fluid-010719-060214. URL: https://doi.org/10.1146/annurev-fluid-010719-060214.

[4] J.L. Castro, C.J. Mantas, and J.M. Benıtez. "Neural networks with a continuous squashing function in the output are universal approximators". In: *Neural Networks* 13.6 (2000), pp. 561–563. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(00)00031-9. URL: http://www.sciencedirect.com/science/article/pii/S0893608000000319.

[5] M Hammache and M Gharib. "An experimental study of the parallel and oblique vortex shedding from circular cylinders". In: *Journal of Fluid Mechanics* 232 (1991), pp. 567–590.

[6] Xiaowei Jin et al. *NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations.* 2020. arXiv: 2003.06496 [physics.comp-ph].

[7] Eamonn Keogh and Abdullah Mueen. "Curse of Dimensionality". In: *Encyclopedia of Machine Learning and Data Mining.* Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2017, pp. 314–315. ISBN: 978-1-4899-7687-1. DOI: 10.1007/978-1-4899-7687-1_192. URL: https://doi.org/10.1007/978-1-4899-7687-1_192.

[8] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2017. arXiv: 1412.6980 [cs.LG].

[9] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8245. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[10] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338. ISSN: 0036-8075. DOI: 10.1126/science.aab3050. eprint: https://science.sciencemag.org/content/350/6266/1332.full.pdf. URL: https://science.sciencemag.org/content/350/6266/1332.

[11] Lu Lu et al. *DeepXDE: A deep learning library for solving differential equations.* 2020. arXiv: 1907.04502 [cs.LG].

[12] Rekha M. "The Ascent of Gradient Descent". In: *Clairvoyant* (2019). URL: https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f.

[13] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[14] Pierremtb. "PINNS in TensorFlow 2.0". In: *Github* (2019). URL: https://github.com/pierremtb/PINNs-TF2.0/blob/master/1d-burgers/inf_cont_burgers.py.

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

[16] Maziar Raissi and George Em Karniadakis. "Hidden physics models: Machine learning of nonlinear partial differential equations". In: *Journal of Computational Physics* 357 (2018), pp. 125–141. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2017.11.039`. URL: `http://www.sciencedirect.com/science/article/pii/S0021999117309014`.

[17] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017. arXiv: `1711.10561 [cs.AI]`.

[18] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017. arXiv: `1711.10566 [cs.AI]`.

[19] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017. arXiv: `1711.10566 [cs.AI]`.

[20] Yeonjong Shin. "On the Convergence of Physics Informed Neural Networks for Linear Second-Order Elliptic and Parabolic Type PDEs". In: *Communications in Computational Physics* 28.5 (June 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`.

[21] Christoph Van Der Malsburg. "Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms". In: *Brain theory*. Springer, 1986, pp. 245–248.

[22] Yunhai Xiao, Zengxin Wei, and Zhiguo Wang. "A limited memory BFGS-type method for large-scale unconstrained optimization". In: *Computers Mathematics with Applications* 56.4 (2008), pp. 1001–1009. ISSN: 0898-1221. DOI: `https://doi.org/10.1016/j.camwa.2008.01.028`. URL: `http://www.sciencedirect.com/science/article/pii/S0898122108001028`.