Physics-informed Neural Networks for Harmonic Maps

Author:
Mattia Nino BARBIERE
June 6, 2025

Supervisor:
Prof. Pablo ANTOLIN SANCHEZ
Chair of Numerical Modelling and Simulation
École Polytechnique Fédérale de Lausanne (EPFL)

A project submitted in partial fulfillment
of the requirements for the degree of
Master of Applied Mathematics

École Polytechnique Fédérale de Lausanne
School of Basic Sciences - Mathematics Section
Lausanne, Switzerland

Abstract

This report investigates the use of Physics-Informed Neural Networks (PINNs) to approximate solutions of partial differential equations (PDEs), with a particular focus on harmonic map problems. After experimenting with different PDE formulations and the structure of PINNs, we focus on how to enforce boundary conditions in the training process. Through a series of numerical experiments on classic examples of PDEs, we observe the most effective network architectures, activation functions, and optimization strategies. Lastly, we apply these insights to solve some harmonic map problems, where exact solutions cannot be derived analytically. The results highlight the strengths and limitations of PINNs in capturing geometrically intricate boundary conditions and suggest directions for further improvement in training stability and model generalization.

Contents

1 Introduction

In recent years, Physics-Informed Neural Networks (PINNs) have emerged as a powerful framework for solving partial differential equations (PDEs) by exploiting machine learning techniques that have made deep learning so successful. The key idea is to encode the PDE into the loss function of a neural network, thereby guiding the model to approximate functions that satisfy the mathematical constraints.

This project focuses on the application of PINNs to a specific class of PDEs known as harmonic map equations. Harmonic maps play a central role in computational geometry, mesh generation, and isogeometric analysis, as they provide smooth mappings between domains. The geometric complexity of the target domains and the nonlinearity of the equations make solving the harmonic map problems computationally expensive. PINNs provide a promising alternative by allowing flexible approximations that generalize across problem instances.

A central challenge in this context is imposing boundary conditions, which is a critical step towards obtaining satisfactory solutions. This report investigates two distinct strategies for incorporating boundary conditions into the PINN framework: a strong imposition approach, which enforces the boundary analytically through an embedding layer, and a weak formulation, in which any deviation from the boundary condition is penalized by the loss function.

To establish a baseline, the project begins with simpler nonlinear second-order elliptic PDEs. These can be further subdivided into divergence and non-divergence form. These experiments help with figuring out which architectural choices, activation functions, and optimization algorithms work well within the PINN framework. The insights gained are then applied to a series of harmonic map problems with increasing geometric complexity. In particular, we also considers a parametrized family of boundary conditions to assess the model's ability to generalize.

The report is organized as follows. Section 2 introduces the necessary mathematical background and notation. Section 3 outlines the PINN framework for solving PDEs. Section 4 discusses the treatment of boundary conditions. Section 5 presents experimental results for standard PDEs, while Section 6 focuses on harmonic map problems. Finally, Section 7 outlines potential improvements and directions for future research.

2 Preliminaries

The goal of this section is to present some notation and examples of PDEs that will be used when performing experiments.

2.1 Notation

Throughout this report, we use the following standard differential operators: $D\boldsymbol{u}$ denotes the Jacobian of a vector-valued function $\boldsymbol{u} : \mathbb{R}^n \to \mathbb{R}^n$, and for real-valued functions $u : \mathbb{R}^n \to \mathbb{R}$, and with a slight abuse of notation, Du denotes the gradient. The divergence operator is defined as

$$\nabla \cdot \boldsymbol{u} = \sum_{i=1}^{n} \frac{\partial u_i}{\partial x_i}$$

where x_i and u_i are the i-th coordinate and coordinate function respectively. Next the Hessian of u is the matrix $(D^2u)_{ij} = \frac{\partial^2 u}{\partial x_i \partial x_j}$ and the Laplacian operator is defined as

$$\Delta u = \nabla \cdot (Du) = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2}.$$

Lastly I denote the *Frobenius inner product* between two matrices $A, B \in \mathbb{R}^{n\times n}$ as

$$A : B = \mathrm{Tr}\big(A^T B\big) = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}B_{ij}.$$

When working with functional spaces for any set $\Omega \subset \mathbb{R}^n$ let $L^p(\Omega)$ represent the space of p (Lebesgue) integrable functions and $H^{1/2}(\partial\Omega)$ be the standard trace space.

In this report, we focus exclusively on second-order elliptic PDEs. In particular, this class of PDEs can be expressed in two different forms.

Definition 2.1 (Divergence and Non-Divergence Form PDEs)**.** For some unknown function $u : \mathbb{R}^n \to \mathbb{R}$ we say a partial differential equation is in *divergence form* [1] if it is written as

$$\begin{cases} \nabla \cdot (A(\mathbf{x})Du(\mathbf{x})) = f(\mathbf{x}) & \text{for } \mathbf{x} \in \Omega \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{for } \mathbf{x} \in \partial\Omega \end{cases} \tag{2.1}$$

Conversely, we say a partial differential equation is in *non-divergence form* [2] if it is written as

$$\begin{cases} A(\mathbf{x}) : D^2u(\mathbf{x}) = f(\mathbf{x}) & \text{for } \mathbf{x} \in \Omega \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{for } \mathbf{x} \in \partial\Omega \end{cases} \tag{2.2}$$

for some $\Omega \subset \mathbb{R}^n$, $f \in L^2(\Omega)$, $g \in H^{1/2}(\partial\Omega)$, and $A \in L^\infty(\Omega)^{n\times n}$ a symmetric and uniformly elliptic matrix function. As a naming convention, we call f the *source term*, g the *boundary term* or *boundary condition*, and A the *diffusion matrix*.

Note that these definitions can be extended to include lower order terms in the differential equations for a more general definition.

Remark 2.1. Observe that in the particular case that $A(\boldsymbol{x}) = I_n$ for all $\boldsymbol{x} \in \Omega$ both divergence form and non-divergence form PDEs reduce to an equation with the Laplacian operator on the left hand side.

2.2 Model problems

Let us now see some examples of PDEs that will be used when performing numerical experiments. All the examples will have $n = 2$ and $\Omega = [0, 1]^2$.

Example 2.1 (Eigenfunction Source PDE)**.** Given parameters $a, b, c \in \mathbb{R}$, the eigenfunction PDE is

$$\begin{cases} \Delta u(\boldsymbol{x}) = -c \sin(a\pi x_1) \sin(b\pi x_2) & \text{for } \boldsymbol{x} \in [0, 1]^2 \\ u(\boldsymbol{x}) = 0 & \text{for } \boldsymbol{x} \in \partial([0, 1]^2). \end{cases}$$

It is easy to see that if $a, b \in \mathbb{Z}$ the analytical solution is

$$u(\boldsymbol{x}) = \frac{c}{(a\pi)^2 + (b\pi)^2} \sin(a\pi x_1) \sin(b\pi x_2).$$

Example 2.2 (Constant Source PDE)**.** Take the following PDE

$$\begin{cases} \Delta u(\boldsymbol{x}) = 4 & \text{for } \boldsymbol{x} \in [0, 1]^2 \\ u(\boldsymbol{x}) = x_1^2 + x_2^2 & \text{for } \boldsymbol{x} \in \partial([0, 1]^2). \end{cases}$$

It is easy to see that the analytical solution to this PDE is

$$u(\boldsymbol{x}) = x_1^2 + x_2^2.$$

In light of Remark 2.1 both Example 2.1 and Example 2.2 can be written in divergence form and in non-divergence form with diffusion matrix $A(\boldsymbol{x}) = I_2$ for all $\boldsymbol{x} \in [0, 1]^2$.

Example 2.3 (Non-constant diffusion matrix PDE)**.** Take the following PDE

$$\begin{cases} \nabla \cdot (A(\boldsymbol{x}) Du(\boldsymbol{x})) = f(\boldsymbol{x}) & \text{for } \boldsymbol{x} \in [0,1]^2 \\ u(\boldsymbol{x}) = x_1^2 + x_2^3 & \text{for } \boldsymbol{x} \in \partial([0,1]^2). \end{cases}$$

where

$$A(\boldsymbol{x}) = \begin{pmatrix} x_1 + 2 & x_1 x_2^2 \\ x_1 x_2^2 & x_2 + 3 \end{pmatrix}$$

and

$$f(\boldsymbol{x}) = 3x_2^4 + 9x_2^2 + 4x_1^2 x_2 + 18x_2 + 4x_1 + 4.$$

With some computations it can be shown that the analytical solution is

$$u(\boldsymbol{x}) = x_1^2 + x_2^3.$$

Example 2.4 (Convection Dominated PDE [3])**.** An example of convection dominated problem, inspired from [3], is the following PDE

$$\begin{cases} A(\boldsymbol{x}) : D^2 u(\boldsymbol{x}) = f(\boldsymbol{x}) & \text{for } \boldsymbol{x} \in [0,1]^2 \\ u(\boldsymbol{x}) = \sin((2x_1 - 1)\pi) \sin((2x_2 - 1)\pi) & \text{for } \boldsymbol{x} \in \partial([0,1]^2). \end{cases}$$

where

$$A(\boldsymbol{x}) = \begin{pmatrix} 1 & 0 \\ 0 & a(\boldsymbol{x}) \end{pmatrix} \qquad a(\boldsymbol{x}) = \arctan\big(K((2x_1 - 1)^2 + (2x_2 - 1)^2 - 1)\big) + 2$$

for some parameter $K \in (0, \infty)$. The source term $f(\boldsymbol{x})$ is chosen in such a way that the analytical solution is

$$u(\boldsymbol{x}) = \sin((2x_1 - 1)\pi) \sin((2x_2 - 1)\pi).$$

3 Physics-informed neural networks for solving PDEs

Physics-informed neural networks are a deep learning framework that aims incorporate partial differential equations as part of the neural network architecture and training process. PINNs have become increasingly popular with the advent of deep learning as a way of incorporating physical laws, such as energy conservation, into models that would otherwise lack any understanding of physical limitations. The key idea is to formulate the loss function in such a way as to incentivize the model to obey, as closely as possible, the constraints given by the PDE in question. By appropriately formulating the deviations of the model from the PDE, we could apply gradient descent optimization techniques so that the network learns to obey the PDE.

In this project, the use of PINNs is specifically focused on approximating solutions of PDEs. Nevertheless the approach is very similar. After sampling random points over the domain, the loss function will be a measure of how well the model satisfies the PDE and by using standard machine learning optimization techniques we can reduce this deviation until a satisfactory approximation is achieved.

3.1 Model architecture

A lot of the technical aspects of PINNs for this project were taken from [4] which provides a lot of details on how PINNs should be structured and trained. For the architecture [4] provides some heuristics for network architectures that have demonstrated good empirical performance. Nonetheless, for this project the only architecture that was used was the multi-layer perceptron (MLP) which performed very well in most cases, provided the network was large enough.

3.2 The loss function

The fundamental aspect of training PINNs to solve PDEs is encoding, in a meaningful way, all the necessary constrains into the loss function. In particular, this includes the PDE and any accompanying boundary condition.[1]

In order to formulate the error, and thus the loss function, between a PINN and the PDE we first define the *residual* as the pointwise deviation of our model $\hat{u}_{\boldsymbol{\theta}}$ from the PDE. For a PDE in divergence form the residual is defined as

$$r(\boldsymbol{x};\boldsymbol{\theta}) = \nabla \cdot (A(\boldsymbol{x}) D\hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x})) - f(\boldsymbol{x})$$

and for a non-divergence form PDE the residual is

$$r(\boldsymbol{x};\boldsymbol{\theta}) = A(\boldsymbol{x}) : D^2\hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x}) - f(\boldsymbol{x}).$$

The loss function can then be defined as the mean of the squared residuals across a batch of points.

$$\mathcal{L}_{PDE}(\boldsymbol{\theta}) = \frac{1}{|B|} \sum_{i=1}^{|B|} \left(r(\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \right)^2$$

where $|B|$ is the batch size of the batch $B = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(|B|)}\}$.

In practice the differential operators of the network $\hat{u}_{\boldsymbol{\theta}}$ where computed using PyTorch autograd which then allows us to perform backpropagation over the loss function to improve on the model.

The next step is incorporating the boundary condition in the loss function. This step was of critical importance in solving the harmonic map PDE (see Section 6) and thus a lot of time and effort was spent in making sure the model agreed as closely as possible to the boundary condition. This will be developed further in Section 4.

3.3 Training data

As with all machine learning applications, the data is a major factor in determining how well a model will perform. Fortunately, with the aim of solving PDEs, the simplest data generation is the best choice. The data is collected by randomly sampling points in the domain and, after evaluating the residual at the sampled data, backpropagation and gradient descent can be applied. Given enough random samples the data should adequately represent the full domain and thus the PINN should be able represent a solution over the full domain. For most examples presented in this project every PINN is tasked to solve a single PDE and thus the main goal is to reduce the residual as much as possible in order to reach a valid approximation of the true solution.

3.4 Optimization

A substantial part of this project was also dedicated to understanding the best approach to training PINNs. With inspiration from [4] the main algorithm that is used was the Adam optimizer [5] for training. Some experiments were also performed with SGD, however the performance was very similar to that of the Adam optimizer. A key insight also came from [6] which proposed to use Adam or SGD for the initial epochs of training and then switch to a second-order optimizer for the final epochs. The paper proposed various second-order methods among which the L-BFGS optimizer [7] which was chosen for the experiments. This trick, while being costly, dramatically improved the quality of the approximations.

[1] A similar approach could be used if we had to enforce an initial condition, however this is not relevant for this project.

4 Enforcing boundary conditions

An important aspect of solving PDEs, is ensuring the solution agrees with the boundary conditions dictated by the problem. In order to train PINN we must make sure that it reliably and accurately satisfies the boundary conditions.

4.1 Strongly enforcing the boundary condition

The first strategy in enforcing the boundary condition was proposed in [8], where the authors provide a mathematically rigorous way of enforcing boundary conditions which gives a way of satisfying the boundary conditions exactly. The idea is to encode boundary conditions directly into the architecture of the network by using eigenfunctions of the Laplace-Beltrami operator as an embedding layer. Specifically, given a set of functions $\phi_1, \dots, \phi_m$ that solve the eigenfunction problem on a given domain Ω, we define a vector-valued embedding $\boldsymbol{\Phi}(\boldsymbol{x}) := (\phi_1(\boldsymbol{x}), \dots, \phi_m(\boldsymbol{x}))$.

By construction, this embedding satisfies $\boldsymbol{\Phi}(\boldsymbol{x}) = \mathbf{0}$ for all $\boldsymbol{x} \in \partial\Omega$, since each eigenfunctions are zero on the boundary. As a result, any neural network $\hat{u}_{\boldsymbol{\theta}}$ composed with this embedding $\hat{u}_{\boldsymbol{\theta}}(\boldsymbol{\Phi}(\boldsymbol{x}))$ will output a constant value on the boundary. By simply subtracting away this constant

$$\tilde{u}_{\boldsymbol{\theta}}(\boldsymbol{x}) := \hat{u}_{\boldsymbol{\theta}}(\boldsymbol{\Phi}(\boldsymbol{x})) - \hat{u}_{\boldsymbol{\theta}}(\mathbf{0})$$

we ensure that $\tilde{u}_{\boldsymbol{\theta}}(\boldsymbol{x})$ is exactly zero on $\partial\Omega$. To satisfy a generic constant boundary condition $u(\boldsymbol{x}) = k$ on $\partial\Omega$, we can simply add a lifting term to the model as

$$\tilde{u}_{\boldsymbol{\theta}}(\boldsymbol{x}) := \hat{u}_{\boldsymbol{\theta}}(\boldsymbol{\Phi}(\boldsymbol{x})) - \hat{u}_{\boldsymbol{\theta}}(\mathbf{0}) + k.$$

Since $\boldsymbol{\Phi}(\boldsymbol{x}) = \mathbf{0}$ on the boundary, this guarantees $\tilde{u}_{\boldsymbol{\theta}}(\boldsymbol{x}) = k$ for all $\boldsymbol{x} \in \partial\Omega$, satisfying the condition exactly.

For all examples in this report we take $\Omega = [0, 1]^2$ and thus the eigenfunctions used in $\boldsymbol{\Phi}$ correspond to the solutions of the Laplacian eigenvalue problem described in Example 2.1 where the analytical solution is the products of sine functions with integer frequencies.

The first issue of this approach is that the reasoning above assumes a constant Dirichlet boundary condition which is an unrealistic assumption in most cases. However, remains unclear how to exploit this embedding layer to exactly satisfy non-constant boundary conditions. The next approach is the classical method in which boundary conditions are imposed when solving PINNs.

4.2 Weakly enforcing boundary conditions

The alternative suggestion to imposing the boundary condition is the suggestion given by [4], which consists of adding an extra term to the loss function. This approach is more generalizable and easier to use for training. The loss function term for the boundary condition is

$$\mathcal{L}_{BC}(\boldsymbol{\theta}) = \frac{1}{|B_{\partial\Omega}|} \sum_{i=1}^{|B_{\partial\Omega}|} \left(\hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - g(\boldsymbol{x}^{(i)})\right)^2$$

where $B_{\partial\Omega}$ is a batch of points such that $\boldsymbol{x}^{(i)} \in \partial\Omega$ for all $i \in \{1, \dots, |B_B|\}$. The complete loss function in this case is

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{PDE}(\boldsymbol{\theta}) + \lambda \mathcal{L}_{BC}(\boldsymbol{\theta}) \tag{4.1}$$

where λ is a hyperparameter that measures the importance of the boundary condition term in the loss function. Higher values will incentivize the model to focus more on satisfying the boundary condition.

5 Experiments

A lot of experiments were conducted with the aim of finding the best strategy to solve PDEs using PINNs. For the sake of brevity, only some representative examples are presented in this report. All the code and more examples are available on the GitHub repository.[2]

Remark 5.1. For consistency in the visualization, the error plots will follow the following colour scheme.

- Green: The green line in the plots represent the square root of the loss function $\mathcal{L}(\boldsymbol{\theta})$. This includes both the contribution of $\mathcal{L}_{PDE}(\boldsymbol{\theta})$ and $\mathcal{L}_{BC}(\boldsymbol{\theta})$.
- Red: The red curve is only present if the boundary conditions are imposed weakly as explained in Section 4.2. In this case the red curves represent the square root of the boundary loss $\mathcal{L}_{BC}(\boldsymbol{\theta})$.
- Blue: The blue curves represent the relative error between the model's prediction and the true solution computed over a given batch of points $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(k)}\}$. The relative error is given by

$$\frac{\left(\sum_{i=1}^{k} \left|\hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - u(\boldsymbol{x}^{(i)})\right|^2\right)^{1/2}}{\left(\sum_{i=1}^{k} \left|u(\boldsymbol{x}^{(i)})\right|^2\right)^{1/2}}.$$

  Notice that this quantity can only be computed for PDEs that have a analytical solution.

- Orange: The orange curves represent the relative error between the model's gradient and the analytical gradient. Similarly to before the formula is

$$\frac{\left(\sum_{i=1}^{k} \left|D\hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - Du(\boldsymbol{x}^{(i)})\right|^2\right)^{1/2}}{\left(\sum_{i=1}^{k} \left|Du(\boldsymbol{x}^{(i)})\right|^2\right)^{1/2}}.$$

  where the gradient of the model is computed using automatic differentiation. Again this can only be computed if the analytical solution is known.

5.1 Model architecture

To obtain a meaningful approximation of the solution to a PDE, it is important that the chosen neural network architecture has sufficient representational power. For the experiments, the architecture used was a multi-layer perceptron, with its width and depth experimentally selected. Figure 1 illustrates the average values of errors and losses during the final epochs of training as a function of depth when solving Example 2.2. The figure shows that, for models with width 64, deeper architectures perform better. Similar experiments on different PDEs gave comparable results. A width of 64 also appeared to offer the best trade-off between model size and representational power.

5.2 Activation functions

Another important aspect of the neural network is the activation used between layers to add non-linearity to the model. An important observation is that the standard RELU activation is of no use. This due to the fact that the second derivative of the RELU function is 0 and thus the second derivative of models with RELU activation would vanish. The choice of activation function was thus reduced to: tanh, GELU (Gaussian Error Linear Unit) [9], and sigmoid.

[2] See: https://github.com/MattiaBarbiere/PINNs_for_harmonic_maps.

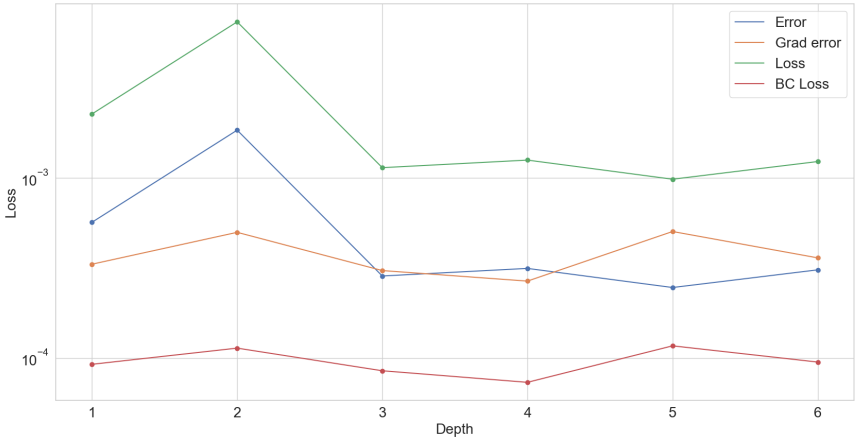
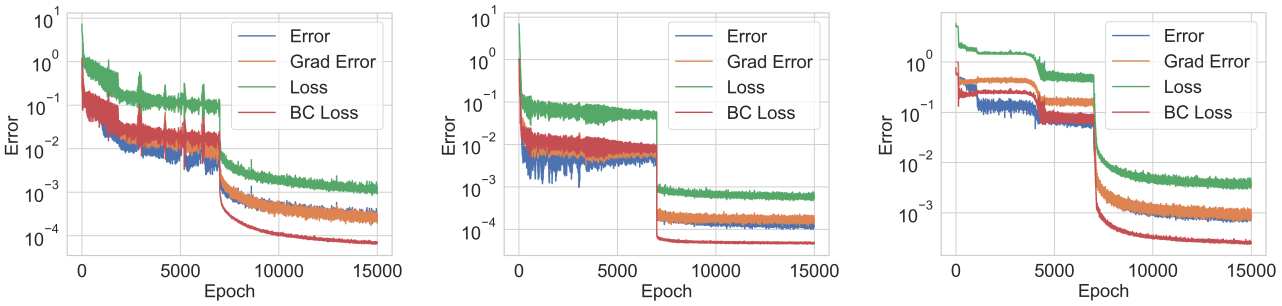


Fig 1. Average values across the 10 *final epochs of losses and errors when training a PINN to solve Example 2.2 as a function of the depth of the model. For this plot the width of the layers is fixed at* 64 *nodes.*



(a) Tanh activation function. *(b) GELU activation function.* *(c) Sigmoid activation function.*

Fig 2. Losses and errors for a PINN solving Example 2.2 with varying activation functions in the model architecture.

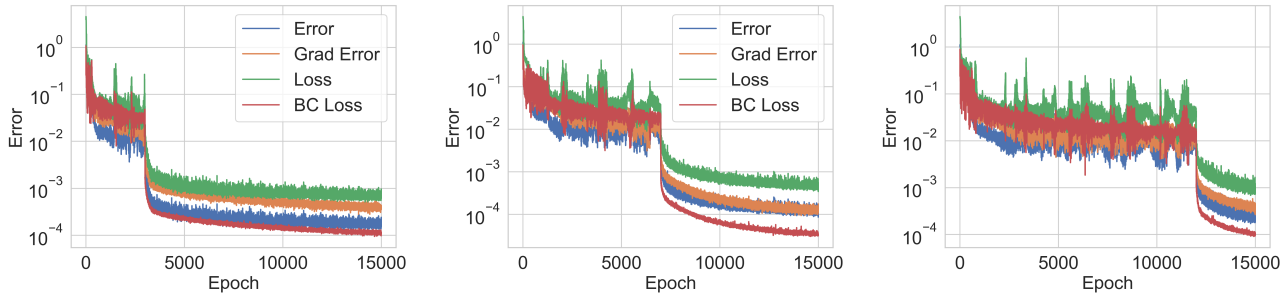
From Figure 2 the obvious choice is GELU as the activation function which reaches a plateau very quickly. This is beneficial as it helps reduce training time. Figure 2 was constructed by solving Example 2.2 however a similar conclusion is made when the same experiment was performed on other PDE problems.

5.3 Training

Following in the footstep of [4] and [6], the best optimization method was using a first-order optimizer for the initial epochs and then move to a second-order optimizer to minimize the loss even further. The intuition is that we use the first optimizer to "explore" the landscape while the second optimizer "exploits" it.

The first step is to choose when to switch from a first-order optimizer to a second-order one. The problem is that the second-order optimizer is about 40 times slower to train than Adam, thus computations with the second-order optimizer should be kept at a minimum. Figure 3 shows the loss function when the PINN was tasked to solve Example 2.3. If we start too early then the computations becomes very expensive and give little improvements in the second half of training. On the other hand, if we delay the switch for too long, the algorithm may not have sufficient time to converge. The compromise between minimizing the loss and reducing computation time was to perform the switch at about half way through the training process.

Further experiments also show that the "exploratory" phase performed by the first-order optimizer is in fact beneficial and leads to an overall better solution.

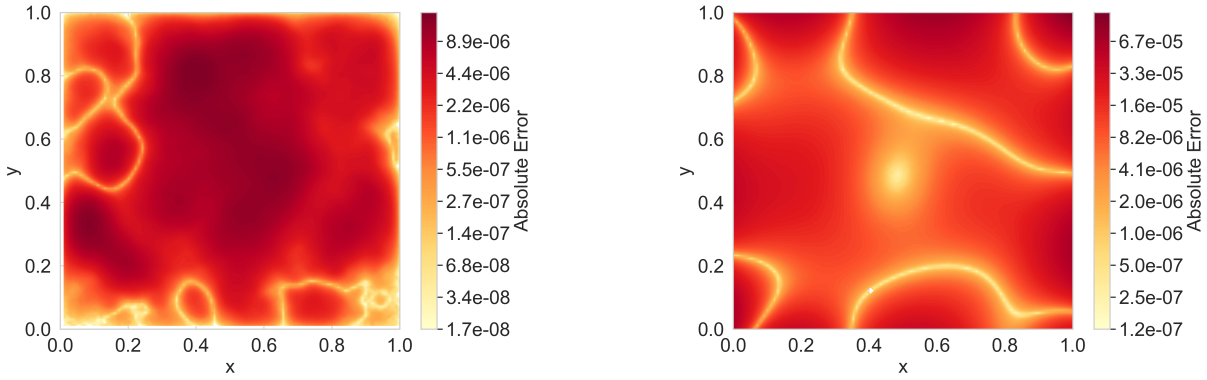


(a) Switch at 3000th epoch *(b) Switch at 7000th epoch* *(c) Switch at 12000th epoch*

Fig 3. The loss functions of the training of a PINN when tasked to solve Example 2.3. The three experiments are identical expect for the value at which the optimizer switches from a first-order method to a second-order one. The training is performed over 15000 *epochs.*

5.4 Strong versus weak enforcement of boundary conditions

To enforce the boundary condition onto the model, strong enforcement is clearly more appealing since, by construction, the model will satisfy the boundary condition with zero error. Figure 4a plots on a log scale the absolute difference between the true solution and the approximated one when solving Example 2.1 when we strongly enforce the boundary condition. As one would expect the approximation is very good close to the boundaries which is represented with the lighter colours. The highest error is thus in the interior of the domain illustrated as darker shades of red. When it comes to the weak enforcement of boundary conditions the situation is reverse. In Figure 4b the areas with the lowest errors are in the interior, while the darker colours appear close to the boundaries.



(a) Strong enforcement. *(b) Weak enforcement.*

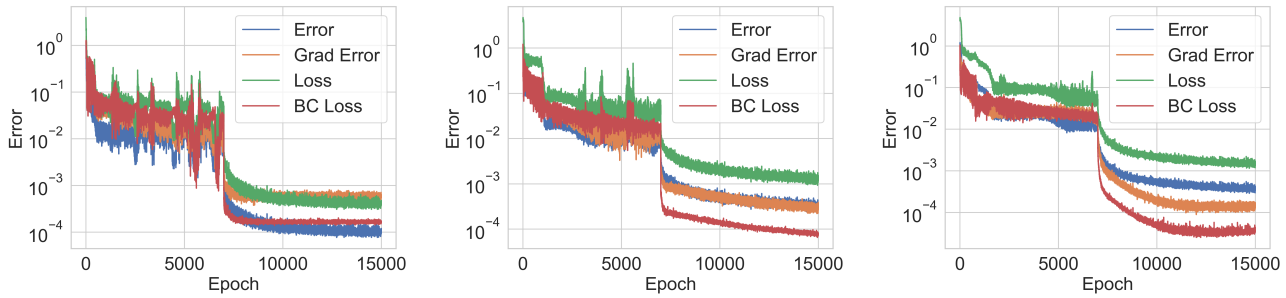
Fig 4. Absolute error in logarithmic scale between the trained PINN and the true solution when trained to solve Example 2.1 with $a = b = c = 1$. *The left image is the absolute error when boundary conditions are enforced strongly and The right image is the absolute error when boundary conditions are enforced weakly.*

As already mentioned the first issue with strongly enforcing boundary conditions is that it only works for constant boundary conditions which is a substantial shortcoming. On the contrary, weakly imposing boundary conditions is very versatile allowing, at least in theory, any arbitrary boundary condition to be approximated.

5.5 Boundary condition weight parameter

Given that weakly enforcing boundary conditions proved to be more useful than strongly enforcement, the next step was choosing the weights λ in the loss function in Equation (4.1). In Figure 5 we notice that by increasing the weight λ, the loss of the boundary condition shown in red does in fact decrease, however once $\lambda > 30$ there is little improvement. Another attempt lower the boundary loss further

was sampling a large amount of points on the boundary of the domain so that the model would be exposed to a much larger sample on the boundary. This nevertheless proved ineffective.



(a) $\lambda = 2$ *(b)* $\lambda = 30$ *(c)* $\lambda = 50$

Fig 5. Losses when training a PINN to solve Example 2.2 with different weights λ *for the loss function in Equation* (4.1).

5.6 Summary

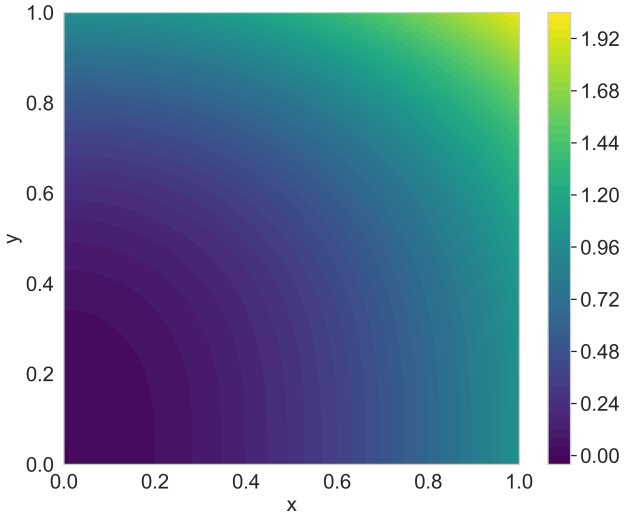
The initial phase of the project revealed many key insights about structuring and training PINNs to solve PDEs. Firstly, a good choice of architecture is the multi-layer perceptron with 6 layers each with 64 nodes. Secondly, the activation function significantly affects the model's ability to learn accurate solutions in as few epochs as possible. The GELU activation function consistently gave the best performance across different PDEs. Additionally, experiments confirmed that the strategy of using first and second-order optimizers during gradient descent works very well, even accounting for the additional computational cost. Moreover, we also performed gradient clipping when optimizing to avoid instabilities in the loss function. Regarding the enforcement of boundary conditions, we saw that weakly imposing boundary conditions with the correct weight comes out on top, by performing well and providing large flexibility. With this knowledge in hand we present some visual examples of solving PDEs using PINNs versus a classic spline benchmark solver.

5.7 Visual examples

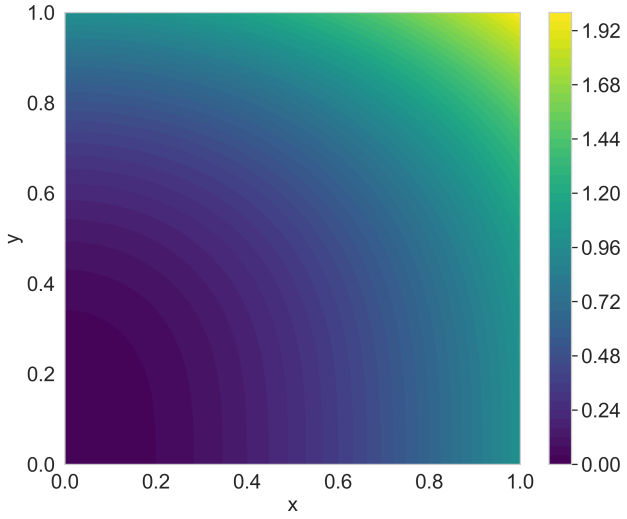
All the examples demonstrate that the PINN framework described above is able to solve a verity of PDEs. Figure 6 shows that the models are able to capture behaviour of the PDE solution even when dealing with complicated diffusion matrices. Furthermore, the losses in Figure 6d show that the model does so with relative ease, since there is little to no sporadic behaviour.

Figure 7 illustrates that PINNs are able to solve PDEs in non-divergence form. The solution given by the trained model is indeed very close to the benchmark solver. Note that comparing the PINN to the analytical solution would give the same conclusion.

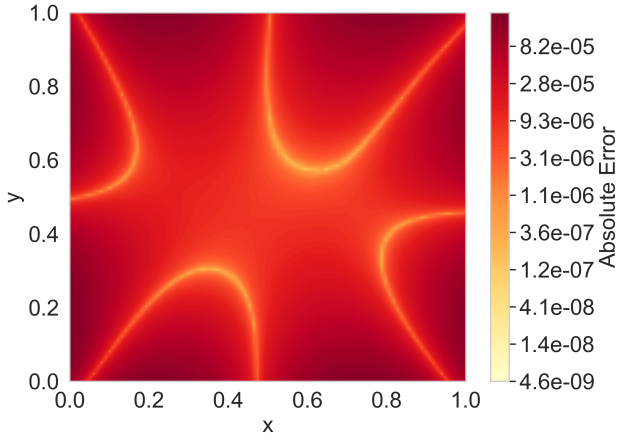
Lastly, Figure 8 shows that the PINN architecture proposed, is has enough representative power to capture rapid oscillations in the solution. It is visible, however, that the PINN does struggle more to solve this problem compared to the others. Indeed, experiments show that if we increase further the frequencies in Example 2.1, the model completely fails to find an adequate solution.



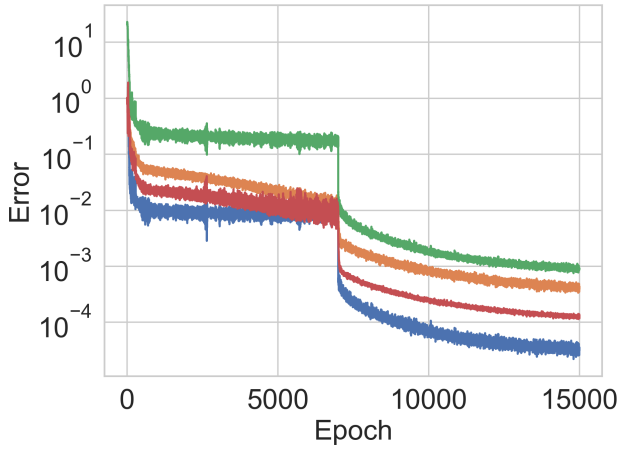
(a) Solution given by a trained PINN solution.



(b) Solution given by the benchmark solver.

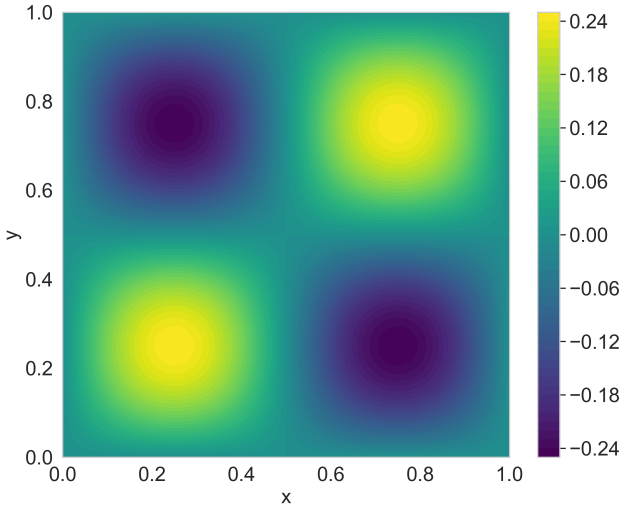


(c) Absolute difference of PINN versus benchmark solution.

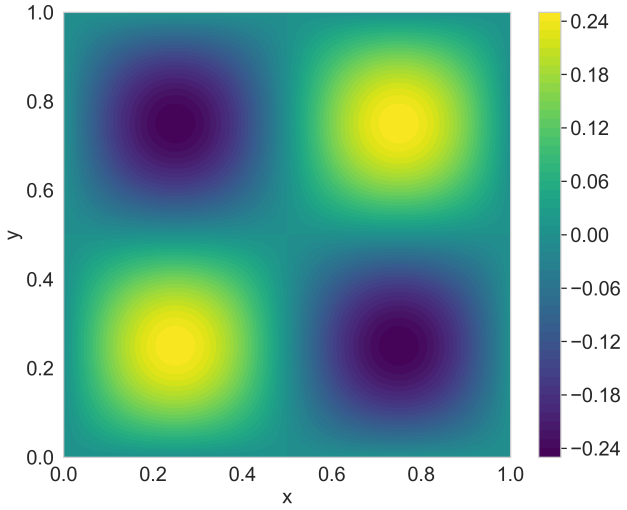


(d) Losses and errors during the training of the PINN.

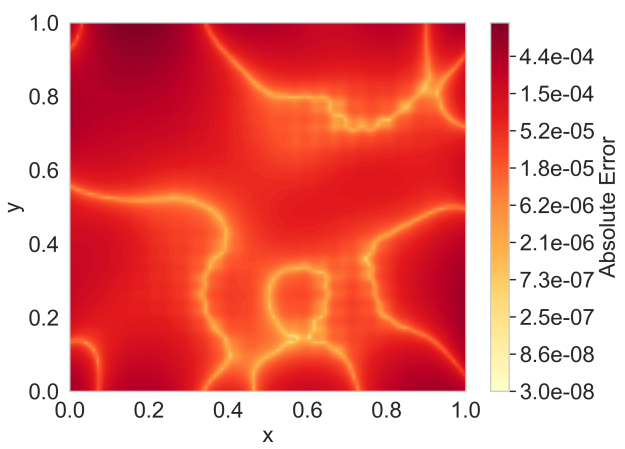
Fig 6. Solving Example 2.3 using a PINN as described in Section 5.6. The first row of images represents the solution given by the trained PINN (left) and the benchmark solver (right). The bottom left image represents the absolute difference on a logarithmic scale between the model and the benchmark solver over the full domain $[0,1]^2$. The bottom right images is the training errors and losses over the training epochs. The colours agree with the convention explain in Remark 5.1.



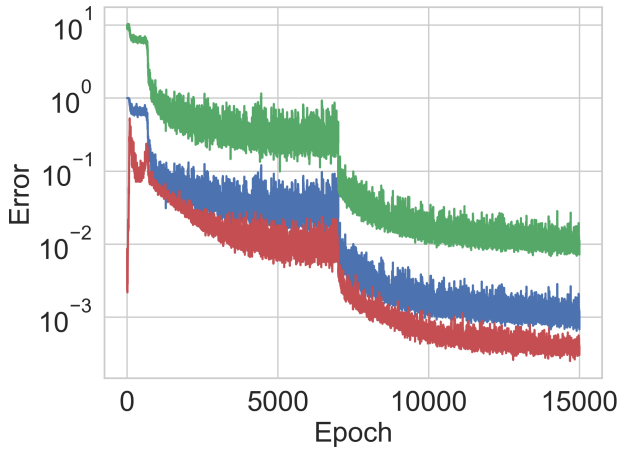
(a) Solution given by a trained PINN solution.



(b) Solution given by the benchmark solver.

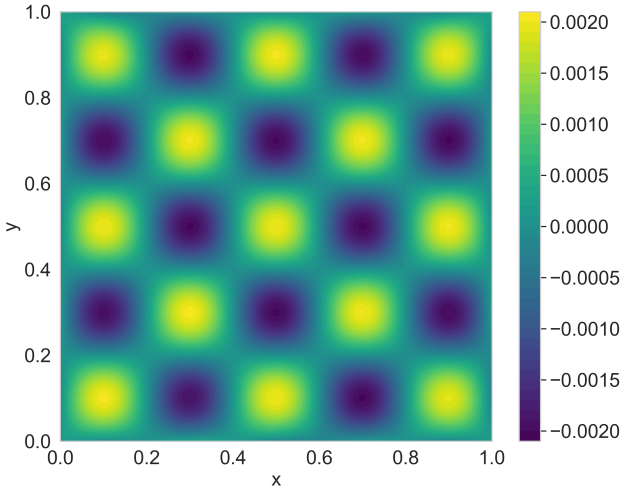


(c) Absolute difference of PINN versus benchmark solution.

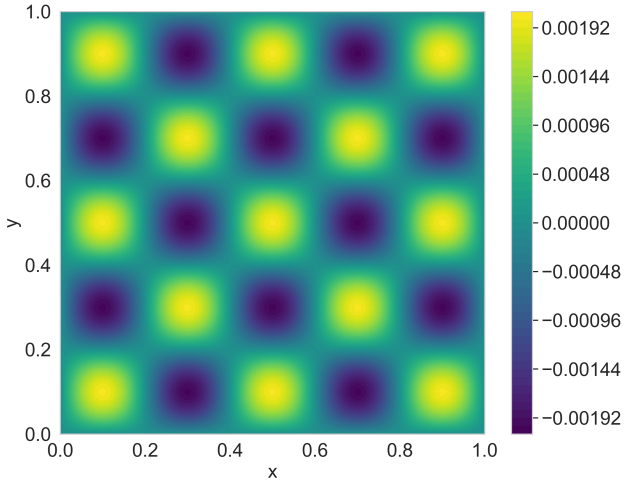


(d) Losses and errors during the training of the PINN.

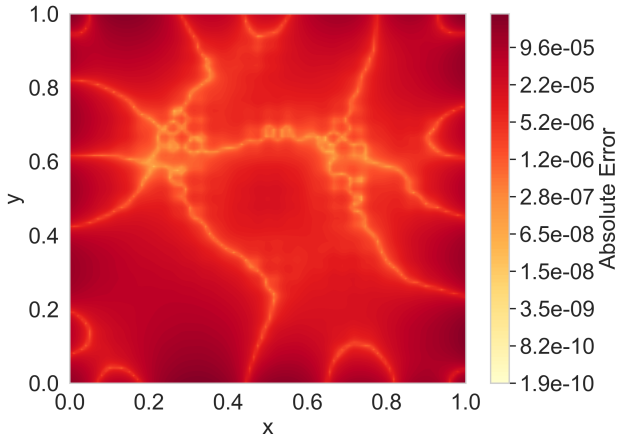
Fig 7. Solving Example 2.4 with $K = 10$ using a PINN as described in Section 5.6. The first row of images represents the solution given by the trained PINN (left) and the benchmark solver (right). The bottom left image represents the absolute difference on a logarithmic scale between the model and the benchmark solver over the full domain $[0, 1]^2$. The bottom right images is the training errors and losses over the training epochs. The colours agree with the convention explain in Remark 5.1.



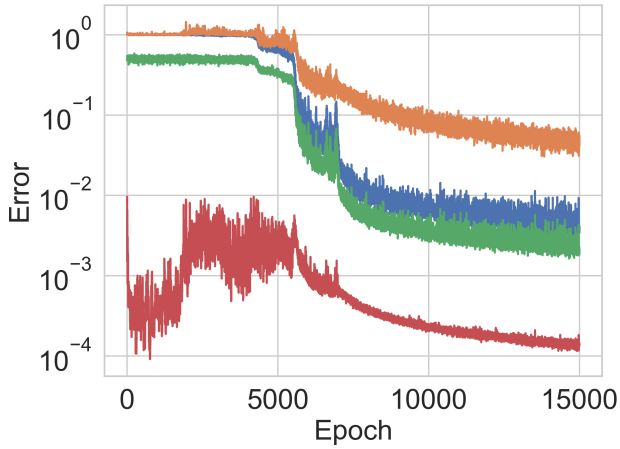
(a) Solution given by a trained PINN solution.



(b) Solution given by the benchmark solver.



(c) Absolute difference of PINN versus benchmark solution.



(d) Losses and errors during the training of the PINN.

Fig 8. Solving Example 2.1 with $a = b = 5$ and $c = 1$ using a PINN as described in Section 5.6. The first row of images represents the solution given by the trained PINN (left) and the benchmark solver (right). The bottom left image represents the absolute difference on a logarithmic scale between the model and the benchmark solver over the full domain $[0, 1]^2$. The bottom right images is the training errors and losses over the training epochs. The colours agree with the convention explain in Remark 5.1.

6 Harmonic map problem

With the key insights gained in the experimental phase of the project, the next step was to solve the full harmonic map problem. The solution to the harmonic map problem can be interpreted as a smooth way to deform one domain into another while preserving geometric structure as much as possible. This mapping provides a way of transforming certain properties of the input domain into properties of the target domain. The harmonic map problem can be mathematically formulated as a quasi-linear variant of Equation (2.2).

Definition 6.1 (Harmonic Map problem [2])**.** Let $\Omega \subset \mathbb{R}^2$. The harmonic map problem consists of finding a vector-valued function $\boldsymbol{u}$ that solves the following system of PDEs

$$A(D\boldsymbol{u}(\boldsymbol{x})) : D^2 u_i(\boldsymbol{x}) = 0 \quad \boldsymbol{x} \in \Omega \quad \text{for} \;\; i \in \{1, 2\}$$

where

$$A(D\boldsymbol{u}(\boldsymbol{x})) = \begin{pmatrix} p_{22} & -p_{12} \\ -p_{12} & p_{11} \end{pmatrix}, \quad p_{ij} = (\partial_{x_i}\boldsymbol{u}(\boldsymbol{x})) \cdot (\partial_{x_j}\boldsymbol{u}(\boldsymbol{x}))$$

with

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}) \quad \boldsymbol{x} \in \partial([0,1]^2)$$

for some boundary function $\boldsymbol{g} : \partial\Omega \to \mathbb{R}^2$.

Notice that the unknown function $\boldsymbol{u}$ is now a vector-valued function. Nonetheless, the approach to solving this problem with PINNs is very similar to before with only slight modification of the loss function as we compute the loss as the sum of the losses of each coordinate function of $\boldsymbol{u}$. The goal is thus to provide the PINN with a boundary function, and through training, the network should produce a result that closely approximates a harmonic map from the Ω square to the target domain defined by the boundary function $\boldsymbol{g}$.

6.1 Model harmonic map problems

Example 6.1 (Quarter annulus harmonic map)**.** The harmonic map that transforms the $[0,1]^2$ square into a quarter annulus with inner radius 1 and outer radius 2 can be formulated as a PDE using Definition 6.1 with

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{pmatrix} (x_1 + 1)\cos\left(\frac{\pi}{2}x_2\right) \\ (x_1 + 1)\sin\left(\frac{\pi}{2}x_2\right) \end{pmatrix} \quad \boldsymbol{x} \in \partial([0,1]^2).$$

A visualization of this boundary condition is shown in Figure 10a.

The next two examples introduce a family of harmonic map problems that depend on specific parameter choices. By adjusting these parameters, a wide range of harmonic map problems can be constructed.

Example 6.2 (Sinusoidal boundary harmonic map)**.** Given parameters c, f_x and f_y, a harmonic map problem with sine functions as boundaries is given the following boundary condition for Definition 6.1

$$\boldsymbol{g}(\boldsymbol{x}) = \begin{pmatrix} (1 - c\sin(f_x \pi x_2))(2x_1 - 1) \\ (1 - c\sin(f_y \pi x_1))(2x_2 - 1) \end{pmatrix} \quad \boldsymbol{x} \in \partial([0,1]^2).$$

Some examples of this boundary are shown in Figure 11 and Appendix A.

The final example of harmonic map problems, unlike Example 6.2, allows for non-symmetric boundary conditions.

Example 6.3 (Polynomial boundary harmonic map)**.** Given parameters $a_{\text{left}}, a_{\text{right}}, b_{\text{bottom}}$, and b_{top}, a harmonic map problem with polynomial functions as boundaries is given by

$$g_1(\boldsymbol{x}) = \begin{cases} x_1 + a_{\text{left}} \cdot p_{\text{left}}(x_2) & \text{if } x_1 = 0 \text{ (left boundary)} \\ x_1 - a_{\text{right}} \cdot p_{\text{right}}(x_2) & \text{if } x_1 = 1 \text{ (right boundary)} \end{cases}$$

$$g_2(\boldsymbol{x}) = \begin{cases} x_2 + b_{\text{bottom}} \cdot p_{\text{bottom}}(x_1) & \text{if } x_2 = 0 \text{ (bottom boundary)} \\ x_2 - b_{\text{top}} \cdot p_{\text{top}}(x_1) & \text{if } x_2 = 1 \text{ (top boundary)} \end{cases}$$

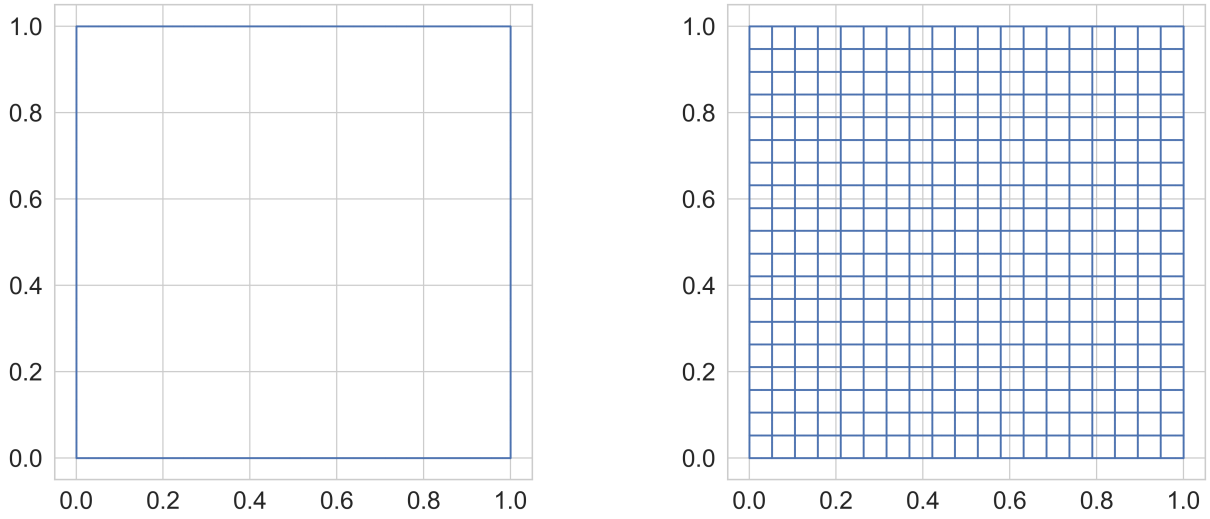
for $\boldsymbol{x} \in \partial([0,1]^2)$ with

$$p_{\text{left}}(x_2) = x_2^2(1-x_2)(1-0.5x_2), \qquad p_{\text{right}}(x_2) = x_2(1-x_2)^2(0.5+x_2),$$
$$p_{\text{bottom}}(x_1) = x_1(1-x_1)^2(1-0.3x_1), \qquad p_{\text{top}}(x_1) = x_1^2(1-x_1)(0.7+x_1)$$

6.2 Experiments

Having learned from the conclusions in Section 5 is comes as a natural to choose the same model as explained in Section 5.6. The training is perform by using Adam for 7000 epochs and then the L-BFGS optimizer for 8000 epochs.

It is difficult to precisely quantify the error between the model and the true solution, as no exact reference solution is available. As such, visualizations are used to illustrate the map learned by the PINN. To do this, a mesh over the domain $[0,1]^2$ (as shown in Figure 9) is passed through the network, producing a deformed mesh over the target geometry which provides a qualitative view of the approximated harmonic map.

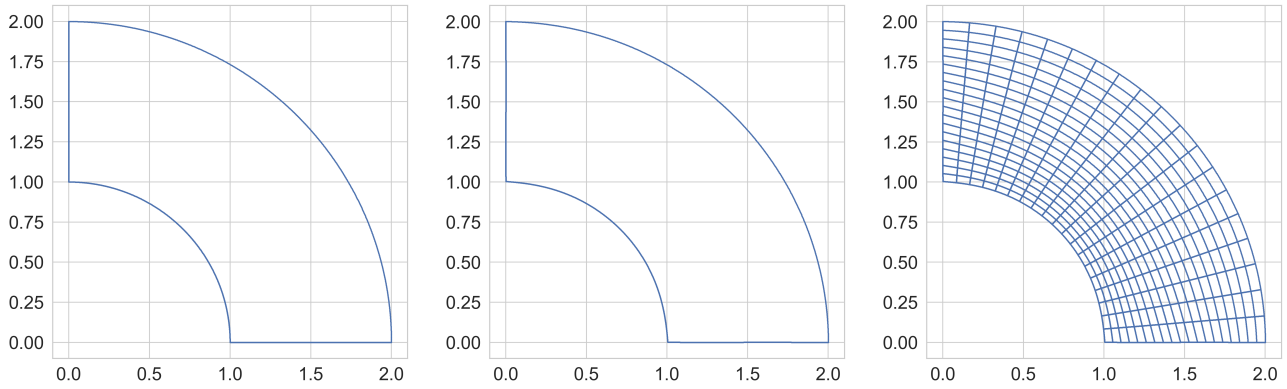


(a) Boundary of the $[0,1]^2$ square. *(b) Mesh over the $[0,1]^2$ square.*

Fig 9. Reference mesh on the $[0,1]^2$ square. Computing a trained model at these shapes provides a visualization for the learned harmonic map.

6.2.1 Quarter annulus harmonic map problem.

The first experiment was mapping the square to a quarter annulus as described in Example 6.1. With a visual inspection of Figure 10 the learned boundary is practically identical to the original boundary. This suggests that the PINN has successfully learned a good approximation of the harmonic map. Indeed, Figure 10c illustrates the harmonic map in action, providing a mapping that transforms a simple mesh on the $[0,1]^2$ square into a mesh on the quarter annulus. This mesh in fact aligns with expectations, as it closely resembles the transformation produced by mapping Cartesian coordinates to polar coordinates.

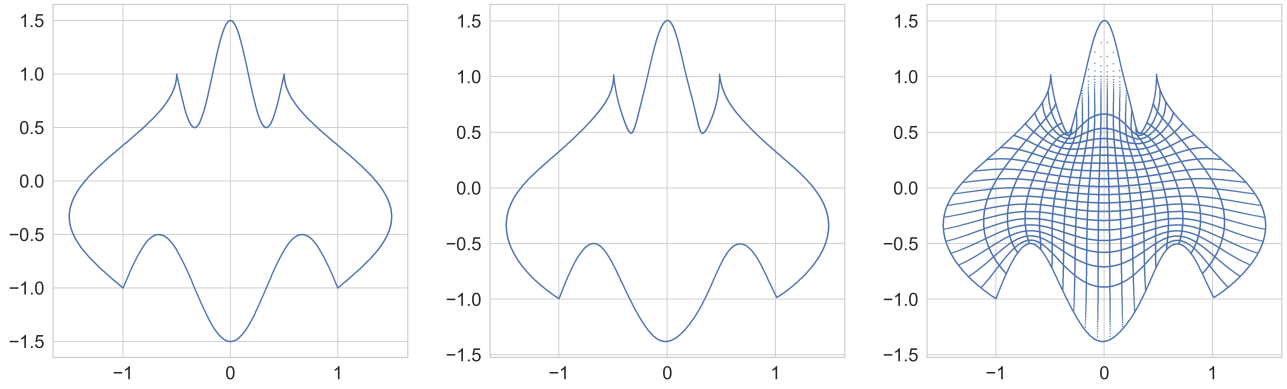


(a) Analytical boundary. *(b) Learned boundary.* *(c) Mesh given by a trained PINN.*

Fig 10. Solving the quarter annulus harmonic map problem Example 6.1. The left image is the analytical boundary given by the boundary condition $\boldsymbol{g}$ *in Example 6.1. The centre figure is the output of the trained PINN when evaluated at the boundary of the* $[0,1]^2$ *square shown in Figure 9a. This represents the boundary that the PINN has learned from training. The right image is the trained PINN evaluated at the mesh of the* $[0,1]^2$ *square shown in Figure 9b.*

6.2.2 Sinusoidal boundary harmonic map problem

The problem in Example 6.2 was constructed to create a wide range of harmonic map problems simply by varying the parameters f_x, f_y, and c. Appendix A provides illustrations for various combinations of parameters. A particularly interesting example occurs when choosing $f_x = -1.5$, $f_y = 3$ and $c = 0.5$ shown in Figure 11a. This example was chosen as it contains both rapid and slow oscillations of the boundary, characteristics that may hinder the model's ability to find an adequate solution. Nevertheless, this is not the case. As presented in Figure 11b and Figure 11c the model does managed to capture the target geometry with only some discrepancies in the bottom portion the the boundary.



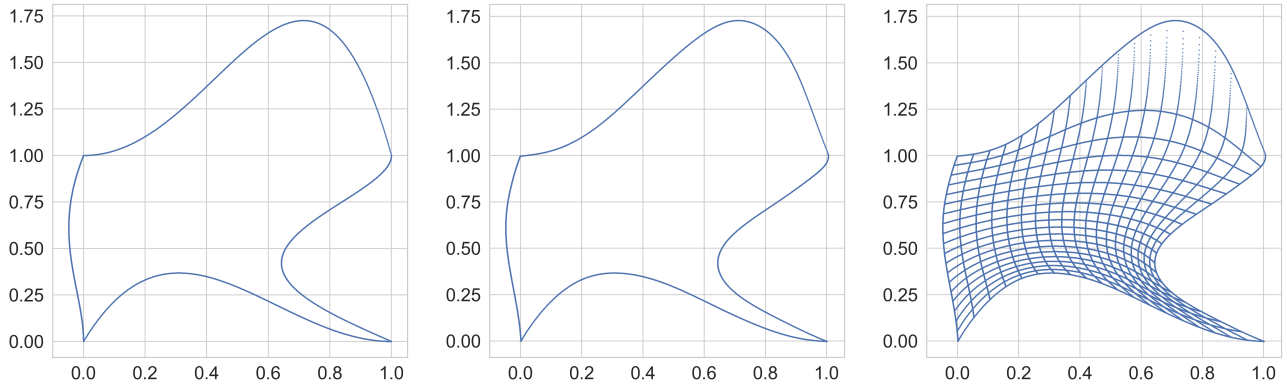
(a) Analytical boundary. *(b) Learned boundary.* *(c) Mesh given by a trained PINN.*

Fig 11. Solving the sine boundary harmonic map problem Example 6.2 with $f_x = -1.5$, $f_y = 3$ *and* $c = 0.5$. *The left image is the analytical boundary given by the boundary condition* $\boldsymbol{g}$ *given by Example 6.3. The centre figure is the output of the trained PINN when evaluated at the boundary of the* $[0,1]^2$ *square shown in Figure 9a. This represents the boundary that the PINN has learned from training. The right image is the trained PINN evaluated at the mesh of the* $[0,1]^2$ *square shown in Figure 9b.*

6.2.3 Polynomial boundary harmonic map problem

Example 6.3 was chosen to allow for harmonic map problems with non-symmetric boundary to be easily constructed. As an example Figure 12 shows the result of a trained PINN on Example 6.3 with parameters $a_{\text{left}} = -0.475, a_{\text{right}} = 2.75, b_{\text{bottom}} = 2.75$, and $b_{\text{top}} = -3.52$. Despite the asymmetry of

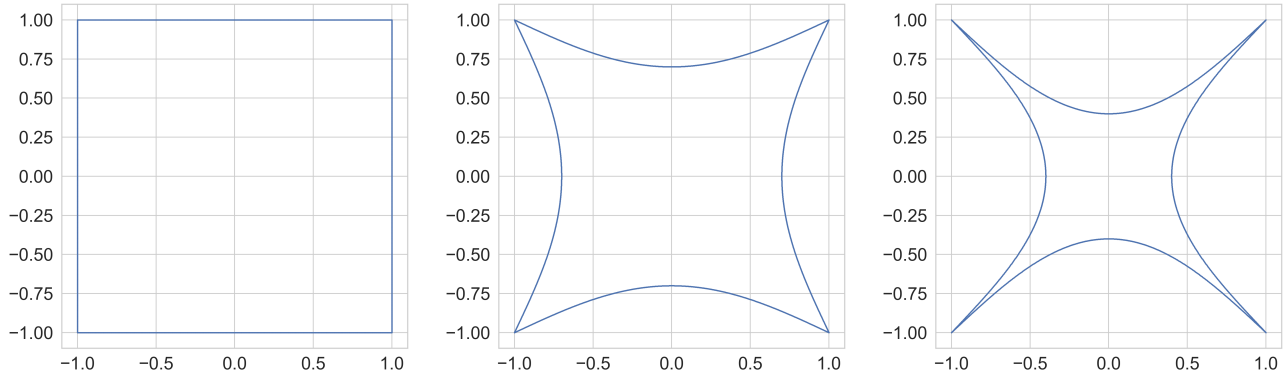
the objective geometry the model still achieves, at least visually, a very good approximation of the boundary, and thus a very good approximation of the harmonic map.



(a) Analytical boundary. *(b) Learned boundary.* *(c) Mesh given by a trained PINN.*

Fig 12. Solving the polynomial boundary harmonic map problem Example 6.3 with $a_{left} = -0.475, a_{right} = 2.75, b_{bottom} = 2.75,$ *and* $b_{top} = -3.52$. *The left image is the analytical boundary given by the boundary condition* $\mathbf{g}$ *given by Example 6.3. The centre figure is the output of the trained PINN when evaluated at the boundary of the* $[0,1]^2$ *square shown in Figure 9a. This represents the boundary that the PINN has learned from training. The right image is the trained PINN evaluated at the mesh of the* $[0,1]^2$ *square shown in Figure 9b.*

6.2.4 Parametrized harmonic map problem

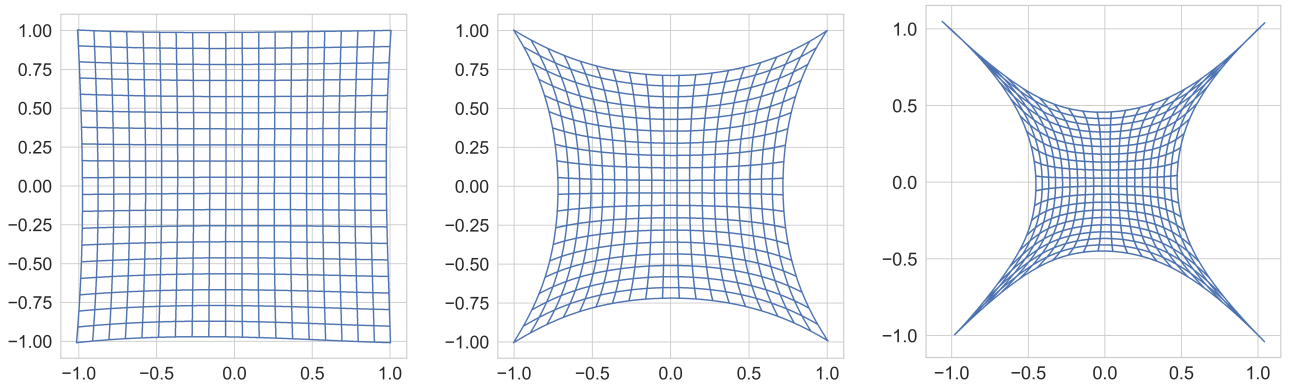


(a) $c = 0$ *(b)* $c = 0.3$ *(c)* $c = 0.6$

Fig 13. Examples of analytical boundaries of Example 6.2 with $f_x = f_y = 1$ *fixed and some choices of* $c \in [0, 0.6]$

While the results illustrated above are very positive, the downside is that the PINN has to be trained from scratch for each PDE we want to solve. The next idea was to check if a single PINN could learn to solve a family of harmonic maps. An example of such a family is given in Example 6.2, where $f_x = f_y = 1$ are fixed and $c \in [0, 0.6]$ is a free parameter. Hence family of harmonic maps is parametrized by c. The idea is for the model to take c^* as input and output the harmonic map identified by $c = c^*$. The analytical boundary conditions for $c \in \{0, 0.3, 0.6\}$ are shown in Figure 13. The training of this model is identical to before with the addition of needing to sample $c \in [0, 0.6]$. Figure 14 illustrates that, once trained, a single model is indeed able to generate meshes for various values of $c \in [0, 0.6]$. This avoids the need of retraining a PINN each time as the value of c is varied.

One case where the model struggles is at $c = 0$, which lies on the boundary of the parameter space. As a result, the model has difficulty accurately fitting the boundary condition due to poor interpolation. Aside from $c = 0$, the model performs remarkably well across all other values of c. Visually, the learned boundaries closely match the true boundaries for the full range of $c \in (0, 0.6]$. Additional results for other values of c are available in Appendix B.

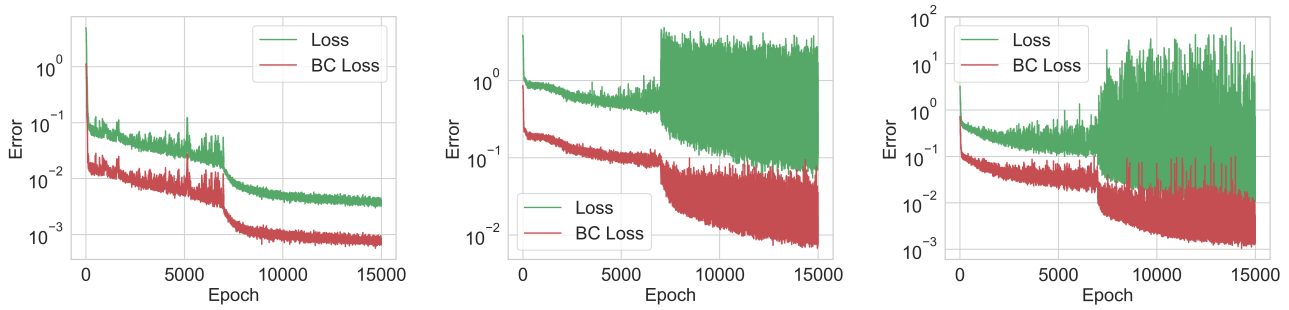


(a) $c = 0$ *(b)* $c = 0.3$ *(c)* $c = 0.6$

Fig 14. Examples of meshes given by a single trained PINN when solving Example 6.2 with $f_x = f_y = 1$ *fixed and* $c \in [0, 0.6]$.

6.2.5 Loss values when solving the harmonic map problem

Despite the promising results presented in the previous sections, the loss function plots shown in Figure 15 indicate that the model struggles during training. Several experiments with the harmonic map problem explained in Section 6.2.2 were performed to identify the cause of this behaviour.



(a) Quarter annulus Section 6.2.1. *(b) Sine boundary Section 6.2.2.* *(c) Polynomial boundary Section 6.2.3.*

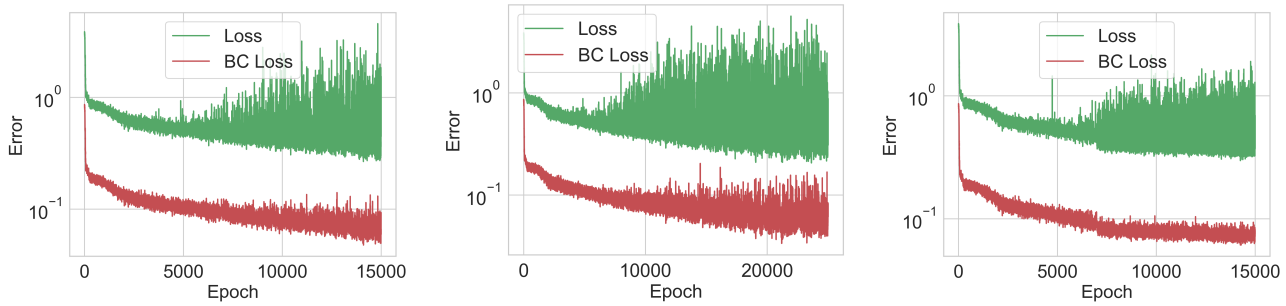
Fig 15. Losses of the PINNs when solving the harmonic map problems. The colours follow the convention of Remark 5.1.

Firstly, the fluctuating behaviour occurs despite applying gradient clipping with a maximum norm of 1 during training. This technique appears to be ineffective, likely because the chosen norm threshold is still too high, allowing relatively large gradients to interfere with the optimization process. Secondly, the fluctuations become more pronounced after switching to a second-order optimizer as shown in Figures 15b and 15c. However, when solving Example 6.2 with $f_x = -1.5$, $f_y = 3$ and $c = 0.5$ using only a first order optimizer as in Figure 16b we also find that the plot exhibits large fluctuations indicating that this behaviour is inherent to the problem.

Another observation was that, unlike Adam, the L-BFGS optimizer uses a fixed learning rate throughout the optimization process. Thus a hypothesis is that the learning rate is too high for the problem. Reducing the learning rate from 1 to 10^{-5} significantly reduces the fluctuations as shown in Figure 16c, however, this also results in a much poorer approximation of the boundary condition, and consequently, of the harmonic map itself.

Ultimately, after further experimentation, we observe that the fluctuations are likely a result of numerical instability. This issue is likely caused by the fact that the loss function already involves computing the Hessian of the model, while the second-order optimizer also computes the second derivatives of the loss function. The increased complexity of the optimization landscape introduced by the harmonic map problem, compared to the PDEs discussed in Section 5, likely amplified inherent numerical instabilities during training. Nevertheless, it should be noted that the second-order optimizer,

despite being more unstable and computationally expensive, achieved a better overall approximation of the harmonic map than using a first-order optimizer alone and thus should not be removed.



(a) Only Adam optimization with 15000 *epochs.*

(b) Only Adam optimization with 25000 *epochs.*

(c) Adam and L-BFGS with a learning rate of 10^{-5}.

Fig 16. Loss values when training a PINN to solve Example 6.2 with $f_x = -1.5$, $f_y = 3$ *and* $c = 0.5$. *The trainings were performed in slightly different ways. In the left image only Adam was used throughout all* 15000 *epochs of training. The centre image is the result of training a PINN using only Adam for* 25000 *epochs. The right image shows the training losses using Adam for* 7000 *epochs followed by* 8000 *epochs with the L-BFGS optimizer with learning rate of* 10^{-5} *compared to a learning rate of* 1 *in Figure 15b.*

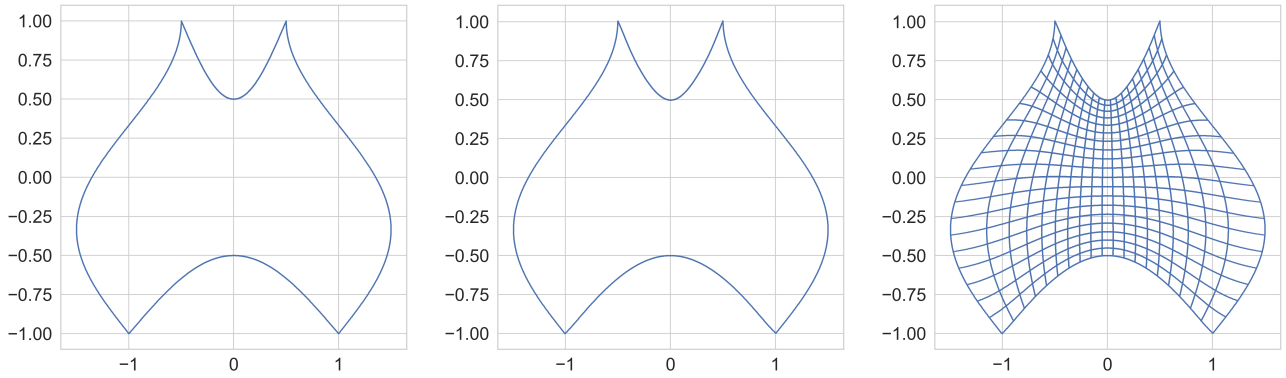
7 Potential improvements and further research

This project presented promising experimental results for using PINNs to approximate solutions of harmonic map problems with complicated boundary conditions. Through many experiments, we identified effective network architectures and activation functions that offered good empirical results. Furthermore, we concentrated on training PINN to satisfy imposed boundary conditions, where we concluded that weakly enforcing boundary conditions worked better.

A key insight, that opens the door to further research, was the successful generalization to parameterized families of harmonic maps in Section 6.2.4, highlighting the potential of a single PINN as solvers for a wide range harmonic map problems.

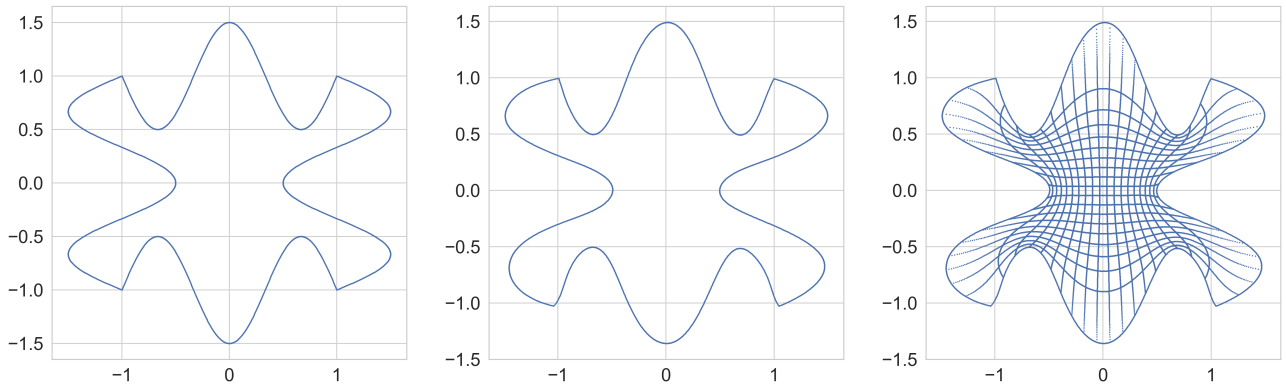
Future research should focus on improving training stability as shown in Section 6.2.5 allowing PINN to find an even better solution. Training a large model and then fine-tuning it to solver generic harmonic problems may also help in reducing the computational cost of solving the harmonic map problem getting us a step closer to general-purpose harmonic map solvers using PINNs.

A Sinusoidal harmonic map examples



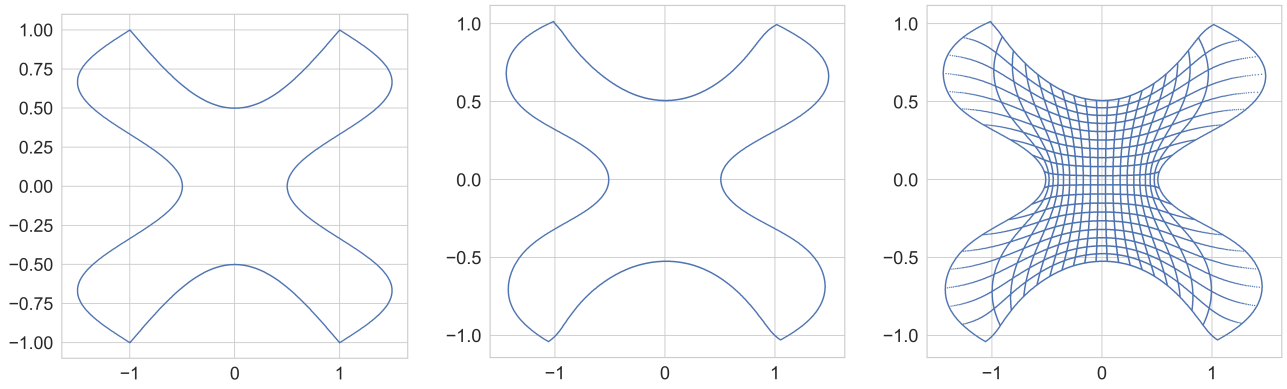
(a) Analytical boundary. *(b) Learned boundary.* *(c) Mesh given by a trained PINN.*

Fig 17. Solving the sine boundary harmonic map problem Example 6.2 with $f_x = -1.5$, $f_y = 1$ and $c = 0.5$. The left image is the analytical boundary given by the boundary condition $\boldsymbol{g}$ given by Example 6.3. The centre figure is the output of the trained PINN when evaluated at the boundary of the $[0,1]^2$ square shown in Figure 9a. This represents the boundary that the PINN has learned from training. The right image is the trained PINN evaluated at the mesh of the $[0,1]^2$ square shown in Figure 9b.



(a) Analytical boundary. *(b) Learned boundary.* *(c) Mesh given by a trained PINN.*

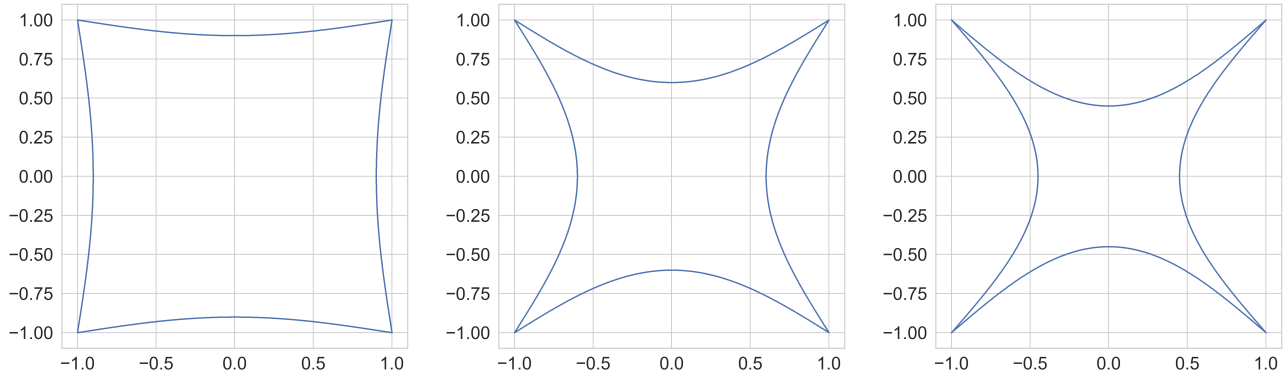
Fig 18. Solving the sine boundary harmonic map problem Example 6.2 with $f_x = -3$, $f_y = 3$ and $c = 0.5$. The left image is the analytical boundary given by the boundary condition $\boldsymbol{g}$ given by Example 6.3. The centre figure is the output of the trained PINN when evaluated at the boundary of the $[0,1]^2$ square shown in Figure 9a. This represents the boundary that the PINN has learned from training. The right image is the trained PINN evaluated at the mesh of the $[0,1]^2$ square shown in Figure 9b.



(a) Analytical boundary. *(b) Learned boundary.* *(c) Mesh given by a trained PINN.*

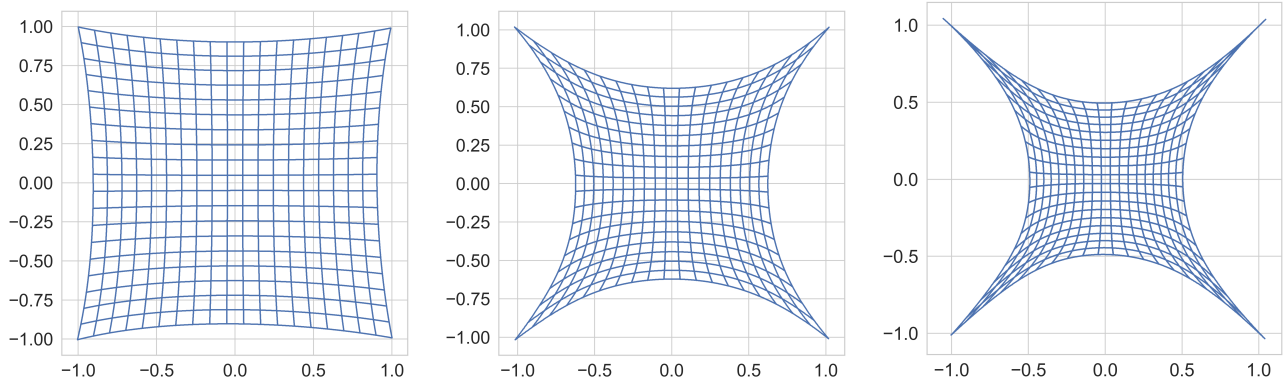
Fig 19. Solving the sine boundary harmonic map problem Example 6.2 with $f_x = -3$, $f_y = 1$ and $c = 0.5$. The left image is the analytical boundary given by the boundary condition $\boldsymbol{g}$ given by Example 6.3. The centre figure is the output of the trained PINN when evaluated at the boundary of the $[0,1]^2$ square shown in Figure 9a. This represents the boundary that the PINN has learned from training. The right image is the trained PINN evaluated at the mesh of the $[0,1]^2$ square shown in Figure 9b.

B Parametrized family of harmonic maps



(a) $c = 0.1$ *(b) $c = 0.4$* *(c) $c = 0.55$*

Fig 20. Examples of analytical boundaries of Example 6.2 with $f_x = f_y = 1$ fixed and some choices of $c \in [0, 0.6]$



(a) $c = 0.1$ *(b) $c = 0.4$* *(c) $c = 0.55$*

Fig 21. Examples of meshes given by a single trained PINN (see Section 6.2.4) when solving Example 6.2 with $f_x = f_y = 1$ fixed and $c \in [0, 0.6]$.

References

[1] D. Gilbarg and N. S. Trudinger, *Elliptic Partial Differential Equations of Second Order* (Classics in Mathematics), en. Berlin, Heidelberg: Springer, 2001, vol. 224. DOI: 10.1007/978-3-642-61798-0. [Online]. Available: http://link.springer.com/10.1007/978-3-642-61798-0 (visited on 05/10/2025).

[2] J. Hinz and A. Buffa, "On the use of elliptic PDEs for the parameterisation of planar multipatch domains," en, *Engineering with Computers*, vol. 40, no. 6, pp. 3735–3764, Dec. 2024, ISSN: 1435-5663. DOI: 10.1007/s00366-024-01997-x. [Online]. Available: https://doi.org/10.1007/s00366-024-01997-x (visited on 05/16/2025).

[3] O. Lakkis and T. Pryer, "A finite element method for second order nonvariational elliptic problems," *SIAM Journal on Scientific Computing*, vol. 33, no. 2, pp. 786–801, Jan. 2011, arXiv:1003.0292 [math], ISSN: 1064-8275, 1095-7197. DOI: 10.1137/100787672. [Online]. Available: http://arxiv.org/abs/1003.0292 (visited on 05/02/2025).

[4] S. Wang, S. Sankaran, H. Wang, and P. Perdikaris, *An Expert's Guide to Training Physics-informed Neural Networks*, arXiv:2308.08468 [cs], Aug. 2023. DOI: 10.48550/arXiv.2308.08468. [Online]. Available: http://arxiv.org/abs/2308.08468 (visited on 05/10/2025).

[5] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs], Jan. 2017. DOI: 10.48550/arXiv.1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980 (visited on 05/26/2025).

[6] E. Kiyani, K. Shukla, J. F. Urbán, J. Darbon, and G. E. Karniadakis, *Which Optimizer Works Best for Physics-Informed Neural Networks and Kolmogorov-Arnold Networks?* arXiv:2501.16371 [cs], Apr. 2025. DOI: 10.48550/arXiv.2501.16371. [Online]. Available: http://arxiv.org/abs/2501.16371 (visited on 04/30/2025).

[7] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," en, *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, Aug. 1989, ISSN: 1436-4646. DOI: 10.1007/BF01589116. [Online]. Available: https://doi.org/10.1007/BF01589116 (visited on 05/26/2025).

[8] M. Kast and J. S. Hesthaven, *Positional Embeddings for Solving PDEs with Evolutional Deep Neural Networks*, arXiv:2308.03461 [math], Aug. 2023. DOI: 10.48550/arXiv.2308.03461. [Online]. Available: http://arxiv.org/abs/2308.03461 (visited on 05/10/2025).

[9] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, arXiv:1606.08415 [cs], Jun. 2023. DOI: 10.48550/arXiv.1606.08415. [Online]. Available: http://arxiv.org/abs/1606.08415 (visited on 05/30/2025).