Available online at www.eccomasproceedia.org
Eccomas Proceedia UNCECOMP (2023) 720-744

ECCOMAS
Proceedia

UNCECOMP 2023
5th ECCOMAS Thematic Conference on
Uncertainty Quantification in Computational Science and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou (eds.)
Athens, Greece, 12-14 June 2023

# QUANTIFYING UNCERTAINTY OF PHYSICS-INFORMED NEURAL NETWORKS FOR CONTINUUM MECHANICS APPLICATIONS

**Damien Bonnet-Eymard[1], Augustin Persoons[1], Matthias G. R. Faes[2] and David Moens[1]**

[1] KU Leuven, Department of Mechanical Engineering, Jan Pieter de Nayerlaan 5, 2860
Sint-Katelijne-Waver, Belgium
address
e-mail: damien.bonnet-eymard@kuleuven.be

[2] TU Dortmund University, Chair for Reliability Engineering, Leonhard-Euler-Strasse 5, 44227
Dortmund, Germany
address
e-mail: matthias.faes [at] tu-dortmund [dot] de

**Abstract.** *Physics-informed neural networks (PINNs) are a relatively new technique that has gained significant attention in recent years as a versatile and robust way to solve a wide range of physical problems, including continuum mechanics. One of the main advantages of using PINNs is that they can directly incorporate physical laws and constraints into the learning process, allowing them to accurately capture the underlying behavior of a system without the need for large amounts of labeled training data. Nevertheless, as they result from a stochastic optimization process, quantifying the model uncertainty is a crucial step before considering real-world applications. In this paper, we study the uncertainty related to the optimization process on PINNs applied to continuum mechanics. We introduce two possible implementations of PINNs to solve boundary value problems (direct and parallel). To be able to quantify the accuracy of the approximations, we address a linear elasticity problem that has an analytical solution. The runs are performed with different neural network architectures and implementations, to quantify the impact of these choices. We found that wide and shallow networks with a Tanh activation function and parallel architecture perform better on average. These results encourage further use of the parallel architecture for inverse quantification of material properties.*

**Keywords:** physics-informed neural networks, uncertainty quantification, continuum mechanics, digital image correlation.

## 1 INTRODUCTION

The use of artificial neural networks (ANNs) to solve physical problems has gained significant attention in recent years [1]. These so-called physics-informed neural networks (PINNs) have indeed shown to be a versatile and robust way to solve a wide range of physical problems, including continuum mechanics [2]. This growing interest may seem surprising, considering the number of already existing and well-established numerical methods for solving physical problems (e.g. finite element method, finite difference method, etc.). However, PINNs have shown great potential in finding the underlying physics of problems by seamlessly combining different sources of information (physical laws, measurements, etc.) into a single model.[3] For continuum mechanics, this can mean finding the material properties from experimental data, with incomplete boundary conditions. This application can be particularly useful with digital image correlation (DIC) measurements, where the displacement field is measured everywhere in the domain (in contrast to the classical strain gauges, where the measurements are only available at specific locations). In this field, the PINN approach can be a valuable alternative to classical techniques (e.g. model updating and virtual fields method) that have their own limitations. [4] [5]

As they result from a stochastic optimization process, PINN models have inherent uncertainty. The implementation of PINNs introduces an important number of parameters (related to the neural network architecture, the optimization process, and the specific problem). As they can have a significant impact on the final results, and understanding the influence of these parameters on the model uncertainty is crucial.

This paper first introduces the PINN approach generally, before focusing on its application to continuum mechanics. Two different PINN architectures are proposed and a list of parameters to consider is given. A specific case of linear elasticity from the literature, where the analytical solution is known, is then studied. The model uncertainty is statistically quantified by performing several optimization runs. The hyperparameters of the PINN are varied to study their impact on the model accuracy and uncertainty. We found that the size of the network, the activation function, and the implementation have a significant impact on the model performance.

## 2 Physics-Informed Neural Networks

In this section, we first contextualize the use of artificial neural networks (ANNs) to solve physical problems, before describing their architectures and how physical laws can be incorporated into the learning process.

### 2.1 Approximate solutions of physical problems with discrete functions

Many physical problems are described by partial differential equations (PDEs). Solving the problem, therefore, means finding a function that satisfies the governing PDE. Most of the time, the solution of a PDE is not known analytically, and discrete functions are used to approximate the real solution. This general approach of searching for an approximate solution inside a finite dimension function space (trial functions) is called the Ritz method (also known as the Galerkin method). The discrete function space is chosen according to some prior knowledge of the solution A famous example of this approach is the finite element method (FEM), where the trial functions are polynomials of a given degree on a specific region of the domain (an element) and equal to zero elsewhere.

Physics-informed neural networks (PINNs) can be considered as a specific case of the Ritz method, where the function space is a neural network. PINNs and FEM have therefore a lot of similarities, as they address the same problem and use the same general approach (for forward problems). A comparison between the two methods is given in table 1.

| | FEM | PINN |
|---|---|---|
| **Trial functions** | Polynomials | Neural networks |
| **Training data localization** | Mesh points | Anywhere in the domain |
| **Optimization** | Least squares | Stochastic optimization |

Table 1: Comparison between the finite element method (FEM) and PINNs.

Whereas in FEM the trial functions are parametrized by the mesh points values (for linear elements), in PINNs the trial functions are parametrized by the neural network weights and biases (see section 2.2). PINN is therefore a mesh-free method, meaning that the training data can be located anywhere in the domain, and not only on the mesh points. However there is no exact solution given a certain discretization (contrary to FEM), and the optimization process is stochastic, leading to potential robustness issues (see section 3).

### 2.2 Artificial neural networks: universal function approximators

Artificial neural networks (ANNs) are a class of machine learning algorithms that are inspired by the structure and function of biological neural networks. ANNs are composed of a set of interconnected units called neurons, which can process information and transmit it to other neurons. The neurons are organized in layers, where the first layer is the input layer, the last layer is the output layer, and the layers in between are called hidden layers. The most simple architecture is a feed-forward neural network (FNN), where each layer is fully connected to the next layer. At each layer, the inputs are multiplied by a weight matrix and added to a bias vector, and then passed through a non-linear activation function :

$$N^i(x) = (\mathbf{W}^i x + \mathbf{b}^i) \quad \forall i \in \{1, \dots, L\} \tag{1}$$

where $x$ is the input vector, $\mathbf{W}^i$ is the weight matrix, $\mathbf{b}^i$ is the bias vector, and $\sigma$ is the activation function. $L$ is the number of layers. This architecture is illustrated in figure 1.

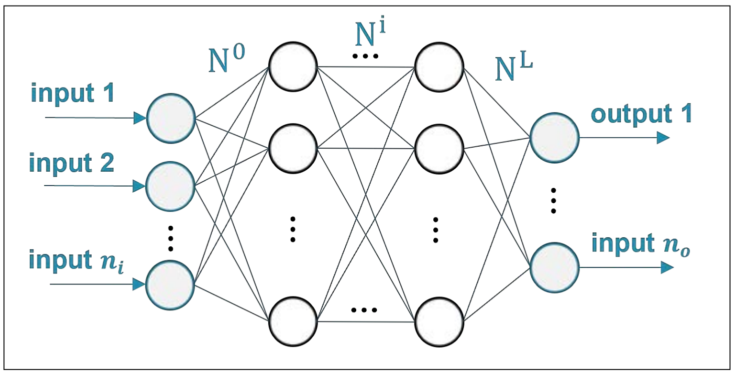

Figure 1: Architecture of a feed-forward neural network.

The width (number of neurons) and depth (number of layers) as well as the activation function are hyperparameters that can be tuned to improve the performance of the ANN.

First introduced in the 1940s by McCulloch and Pitts [6], ANNs gained a lot of popularity in the 1980s thanks to two important developments:

- the development of the backpropagation algorithm [7], which allows to train ANNs using gradient descent
- mathematical proofs that ANNs are universal function approximators [8],[9]

These two properties provide the mathematical background and the practical tools that make ANNs versatile and powerful approximators. The training process consists in finding the weights and biases of the neural network that minimize a loss function (that quantifies the quality of the approximation), which is done using stochastic optimization algorithm (Adam [10] and LBFGS [11] are two popular optimization algorithms). This training process introduces new hyperparameters, such as the optimization algorithm, the learning rate, the batch size, the number of epochs, etc. All these hyperparameters will be later considered to improve the reliability of PINNs (see section 3).

However, this training process generally requires a large amount of data to be efficient, which is usually not the case for physical problems (the number of measurements is limited). The PINN approach overcomes this difficulty by injecting physical laws into the training process (see section 2).

## 2.3 Physics-informed neural networks: neural networks that comply with physical laws

Although the use of ANNs to solve PDEs can be traced back to the 1990s [12], this approach gained a lot of popularity after a paper published by Raissi et al. [13] that first coined the term PINNs for physics-informed neural networks. The main idea of PINNs is to make the neural network comply with some physics prior by adding extra loss terms to the loss function. We illustrate this principle on the following boundary value problem (BVP) :

$$\begin{cases} F(u, \nabla u, \nabla^2 u, \dots) = 0 & \text{in } \Omega \\ u = u_{BC} & \text{on } \partial\Omega \\ u = u_{IC} & \text{at } t = 0 \end{cases} \tag{2}$$

where $\Omega$ is the domain, $u$ is the unknown function, $u_{BC}$ and $u_{IC}$ are the boundary and initial conditions, and $F$ is the PDE depending on $u$ and its derivatives. To solve this problem, a PINN approximation should respect both the PDE and the boundary condition.

### PDE constraint

The PDE constraint is enforced by adding a loss term to the loss function. The PDE is calculated using automatic differentiation (chain rule through the neural network) and the loss term is usually the squared norm of the PDE residual :

$$\mathcal{L}_{\text{PDE}} = \left\| F(u, \nabla u, \nabla^2 u, \dots) \right\|^2 \tag{3}$$

Another possible approach is to use an energy form $\mathcal{E}$ derived from the PDE (variational formulation) :

$$\mathcal{L}_{\text{E}} = \mathcal{E}(u, \nabla u, \nabla^2 u, \dots) \tag{4}$$

### Boundary and initial conditions

The boundary/initial conditions can be enforced like the PDE one, by adding a loss term to the loss function. The prediction of the neural network on the boundary/at time $t = 0$ is compared to the boundary/initial condition, and the loss term is usually the squared norm of the difference :

$$\mathcal{L}_{\text{BC/IC}} = \left\| u_{BC/IC} - u \right\|^2 \tag{5}$$

This approach is called soft constraint because the boundary/initial conditions are not strictly enforced but only penalized. Boundary/initial conditions can also be hardly enforced by multiplying the output of the neural network by a mask function that assures ad hoc compliance with the boundary/initial conditions. As an example, we can consider the following 1D problem :

$$\begin{cases} F(u, \nabla u, \nabla^2 u, \dots) = 0 & \text{in } \Omega = [0, 1] \\ u(0) = u(1) = 0 \end{cases} \tag{6}$$

The following mask function can be used to enforce the boundary conditions on a neural network $\mathcal{N}$ :

$$\mathcal{M}(\mathcal{N}(x)) = x \times (1 - x) \times \mathcal{N}(x) \tag{7}$$

By limiting the number of loss terms, this hard constraint approach can make the training process more efficient but requires finding a mask function that complies with the specific boundary/initial conditions of the problem. Although it may seem limiting, finding a mask function seems to generalize well to domains with complex geometry. [14]

### Data constraint

If some data is available inside the domain (measurements), it can be used as a residual loss to train the neural network :

$$\mathcal{L}_{\text{data}} = \|u_{data} - u\|^2 \tag{8}$$

### Total loss function

The loss terms can be combined to form the final loss function :

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \alpha_{\text{BC/IC}} \times \mathcal{L}_{\text{BC/IC}} + \alpha_{\text{data}} \times \mathcal{L}_{\text{data}} \tag{9}$$

Each loss term can eventually be weighted using a hyperparameter $\alpha$ to control the importance of each loss term in the training process. If the PINN is trained successfully (i.e. the loss function is closed to zero), the neural network approximation should comply with the PDE, the boundary/initial conditions and the ground truth data.

### Forward and inverse problems

The PINN approach can be used to solve both forward and inverse problems depending on the presence or not of ground truth (measurements) data.

In the case of a forward problem, the neural network is trained using only the PDE loss and BC/IC constraints (data loss can be added but is not necessary). The PINN is then very similar to FEM as it finds an approximate solution given a PDE and boundary conditions.
In the case of an inverse problem, the neural network is trained using the PDE loss, BC/IC constraints and data loss. Here some physical parameters (such as Young's modulus for elasticity) of the problem can be trained at the same time as the ANN weights and biases.

PINNs usually bring real added value for inverse problems. On forward problems, PINNs compete with FEM (among other methods) and require a training phase that can be time-consuming compared to matrix inversion in FEM. On inverse problems, PINNs show a great capacity to find underlying physics [3], specifically on ill-posed problems that are difficult to solve with model updating approaches. A common practice is to validate the PINN approximation on a forward problem (compared to FEM) before using it for a more valuable inverse problem.

An overview of the PINN architecture is given in figure 2.

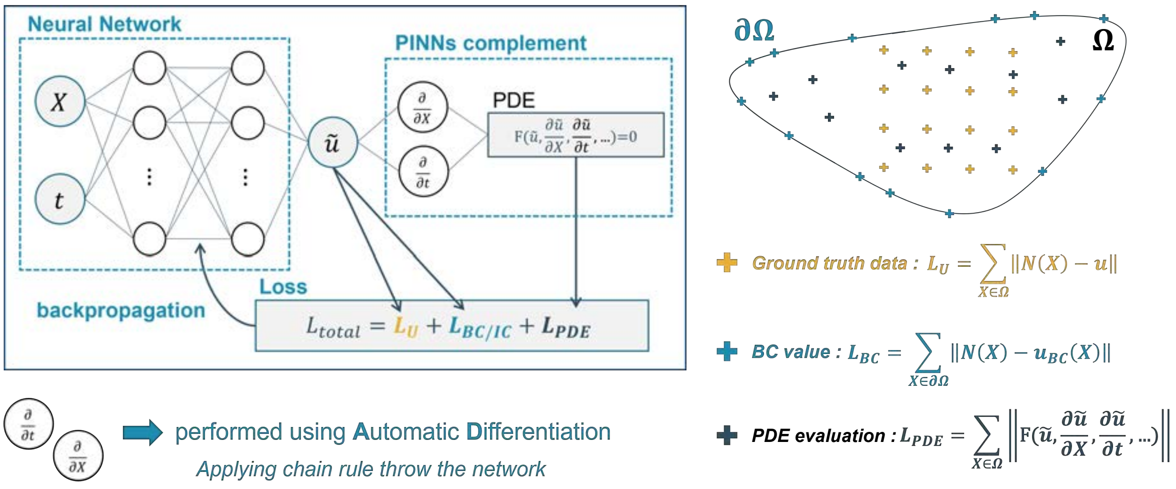
Neural Network
X
t
PINNs complement
PDE
$F(\tilde{u}, \frac{\partial \tilde{u}}{\partial X}, \frac{\partial \tilde{u}}{\partial t}, ...)=0$
$\tilde{u}$
Loss
backpropagation
$L_{total} = L_U + L_{BC/IC} + L_{PDE}$
$\partial\Omega$
$\Omega$
Ground truth data : $L_U = \sum_{X\in\Omega} \|N(X) - u\|$
BC value : $L_{BC} = \sum_{X\in\partial\Omega} \|N(X) - u_{BC}(X)\|$
PDE evaluation : $L_{PDE} = \sum_{X\in\Omega} \left\|F(\tilde{u}, \frac{\partial \tilde{u}}{\partial X}, \frac{\partial \tilde{u}}{\partial t}, ...)\right\|$
performed using Automatic Differentiation
Applying chain rule throw the network

Figure 2: Architecture of a PINN. The neural network is trained to minimize the PDE loss, the BC/IC constraints and the data loss.

## 3 Accuracy of PINNs for continuum mechanics

As mentioned in the previous section, the PINN approximation is the result of a stochastic training process. This stochasticity induces some uncertainty in the PINN model, as several runs of the training process can lead to different PINN approximations. The main objective of this section is to evaluate and enhance the accuracy of the PINNs used to model continuum mechanics problems.

### 3.1 PINNs for continuum mechanics: hyperparameters to consider

Applying PINNs to continuum mechanics problems requires some choice of architecture and hyperparameters that can have a significant impact on the reliability of the approximation.

#### PINN applied to continuum mechanics

In this study, we focus on static boundary value problems in continuum mechanics. Specifically, we aim to determine the displacement $u$ and stress field $\sigma$ within a given domain $\Omega$, subject to known boundary conditions on $\partial\Omega$. The solution to this problem must satisfy the governing equations of continuum mechanics.

The first equation is the kinematic relation between displacement and strain $\epsilon$, which describes the deformation of the material. In the case of small deformations, this relation can be written as :

$$\epsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \qquad \textit{small-deformation assumption} \tag{10}$$

where $u_i, j$ is the partial derivative of the i-th component of the displacement field u with respect to the j-th coordinate (Einstein notation)

The second equation is Newton's second law, which describes the balance of forces within the material. In static situations, this law can be expressed in terms of the stress field $\sigma$ and the volumetric forces $f$ as :

$$\sigma_{ij,j} + f_i = 0 \qquad \textit{momentum balance} \tag{11}$$

In addition to these two equations, a third equation is required to fully describe the material behavior. For our purposes, we assume the material to be homogeneous, isotropic, and following a linear elastic behavior. In this case, the relation between the strain $\epsilon$ and the stress $\sigma$ can be written as :

$$\sigma_{ij} = \lambda\delta_{ij}\epsilon_{kk} + 2\mu \qquad \textit{linear elasticity} \tag{12}$$

Where $\lambda$ and $\mu$ are the Lamé coefficients that parameterize the elastic relation.

Using these equations, it is possible to determine the stress field corresponding to any given displacement field and calculate the corresponding momentum balance. The PDE loss and corresponding potential energy can then be written as follows:

$$\mathcal{L}_{\text{PDE}} = \|\sigma_{ij,j} + f_i\|^2 \tag{13}$$

$$\mathcal{L}_{\text{E}} = \frac{1}{2}\sigma_{ij}\epsilon_{ij} - f_i u_i \tag{14}$$

**Different possible implementations of PINNs for continuum mechanics**

In a continuum mechanics problem, three fields ($u$, $\sigma$, and $\epsilon$) are to be determined. Therefore different implementations are possible, depending on which fields are chosen as the outputs of the neural network. We focus on two implementations (direct and parallel) that are illustrated in Figure 3:

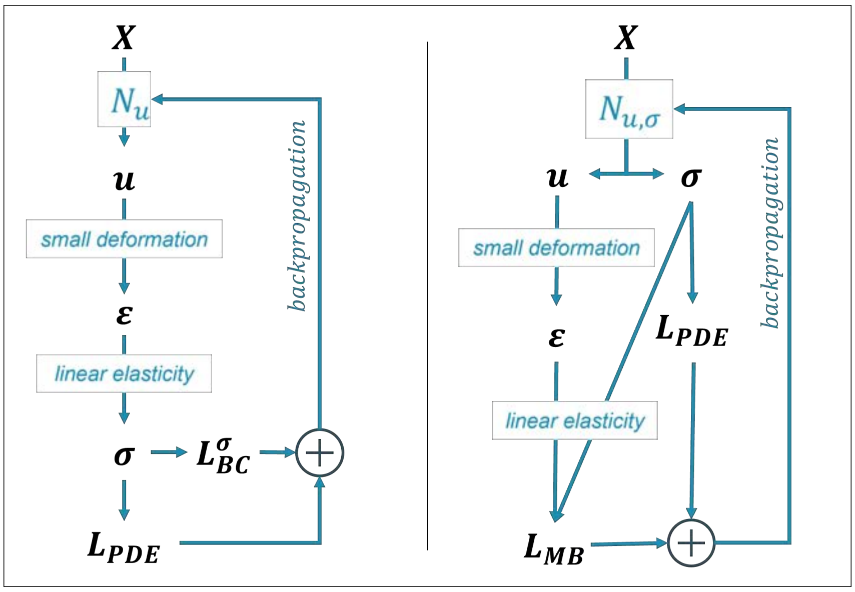

Figure 3: Two possible implementations of PINNs for continuum mechanics: direct (left) and parallel (right).

For the parallel implementation, the material behavior is not respected at first (the stress field is not calculated from the displacement field). The material behavior is then enforced softly by adding a corresponding term to the loss function :

$$\mathcal{L}_{\text{MB}} = \|\sigma_{ij} - \lambda\epsilon_{ij}\delta ij + 2\mu\epsilon_{ij}\|^2 \tag{15}$$

It is important to note that hard constraints on the boundary/initial conditions can only be applied if the constrained field is the output of a neural network. Thus for mixed boundary conditions (e.g. on both $u$ and $\sigma$) in direct implementation, the constraint on $\sigma$ must be enforced softly.

The two implementations, therefore, seem comparable as they have the same amount of soft and hard constraints (see Table 2).

| **Constraints** | **Direct** | **Parallel** |
|---|---|---|
| **BC on** $u$ | Hard | Hard |
| **BC on** $\sigma$ | Soft | Hard |
| **Material behavior** | Hard | Soft |

Table 2: Constraints in the direct and parallel implementations of PINNs for continuum mechanics.

Having these architectures in mind, it is possible to make a list of hyperparameters to consider when applying PINNs to continuum mechanics problems.

## Hyperparameters to consider

The usuals paramaters of machine learning as well as the problem specifics ones are listed in table 3. These hyperparameters can be divided into three categories: the network architecture, the training process and the problem specifics. They will be considered as tunable hyperparameters to improve the reliability of the PINN approximation in the following section.

| | Parameter | Possible values | Comment |
|---|---|---|---|
| Network architecture | Number of layer (depth) | 3, 4, 5 (for example) | Usually, the depth and width are changed simultaneously in order to keep the total number of parameters constant. This makes the models more comparable as they have the same capacity. We generally try to minimize the total number of parameters while keeping a sufficient expressiveness of the neural network (low approximation error), as fewer parameters allow faster convergence and limit over-fitting |
| | Number of neurons per layer (width) | 30, 40, 50 (for example) | |
| | Activation function | Sigmoid, Tanh, ReLU... | The three cited activation functions are the most used in the literature. ReLU is not recommended in [15] as it has discontinuous derivatives and PINNs rely on accurate evaluation of derivatives. |
| | Other architecture | PFNN, ResNet, Transformer... | More advanced architecture than feed-forward can be used. For example parallel neural network (a different ANN for each output) is used in [16] |
| Optimizer | Optimizer type | Adam, LBFG-S... | Several optimizers can also be used one after another during the process. For example authors of the DeepXDE library [17] recommand to start using Adam and finish the training with LBFG-S |
| | Learnin rate | [1e-5 ; 1e-2] (for example) | For gradient descent algorithm (SGD, Adam), choosing the learning rate is a trade-off : it needs to be large enough to have fast convergance at the beginning of the training but small enough to approximate precisely the solution at the end of the training. Learning strategies address this issue by decreasing the learning rate during the training, the name of the strategy corresponds to the decrease pace (stepwise, exponential, cosinus...) |
| | Learning rate strategy | Constant, Step, Cosinus... | |
| | Number of epochs (total iterations) | example: 10000 | The total number of iterations, with the total number of parameters, is linked to the computational cost allocated to each run. It's important to note that this value is an upper limit that may not be reached if a convergence criterion is reached before (LBFG-S has such a criterion). |
| Continuum mechanics | Outputs of the neural network | u, $\sigma$ or both | As explained previously choosing the outputs of the ANN has an impact on how constraints can be imposed. Hard constraints are generally preferred since they ensure exact compliance with the condition without adding a term to the loss. This is why we have retained two possible architectures (direct and parallel) which each contain only one soft constraint. It is also possible to vary only the way a constraint is imposed (soft or hard) in order to evaluate the impact of this choice on the accuracy and reliability of the model. |
| | Boundary/initial condition constraint | soft, hard | |
| | Material behavior constraint | soft, hard | |
| | Physic constraint | PDE, Energy | Whereas the original PINN approach use the PDE as a physic loss, an energy form may be more regular and thus allow better convergence. |
| | Number of residual points | 1000 (for example) | A number of residual points and a sampling strategy must be defined for each loss term (PDE and soft constraint). The total number of residual points (batch size) affects the computational cost. The residual points can be resampled during the training with a potentially evolving sampling strategy (to avoid sampling specific over-fitted solutions). That's the case with Residual Adaptative Distribution (RAD) where the sampling distribution favors high-loss regions.[18] |
| | Sampling strategy | Grid, Random, Sobol, RAD | |

Table 3: Commented hyperparameters of PINN implementation on continuum mechanics problems.

## 3.2 Accuracy of PINNs on a case study of the literature

One of the first implementations of PINN in continuum mechanics was done by Haghighat et al. [16]. They introduced a simple linear elastic problem that has an analytical solution to validate their approach. The code from the original paper is publicly available on GitHub[1] (SciANN-SolidMechanics).

We will address the same problem in this section, to assess the accuracy of the implementation proposed in [16] before trying to improve it through hyperparameter optimization (HPO).

### Problem description

The boundary value problem is defined on the unit square domain: $\Omega = [0, 1]^2$, considering displacement in two dimensions: $u = (u_x, u_y)$. Thanks to the symmetry of the strain and stress tensors ($\epsilon_{xy} = \epsilon_{yx}$ and $\sigma_{xy} = \sigma_{yx}$), they can be written as follows:

$$\epsilon = \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{xy} \end{pmatrix} \quad ; \quad \sigma = \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix}$$

The mixed boundary conditions are summarized in figure 4:

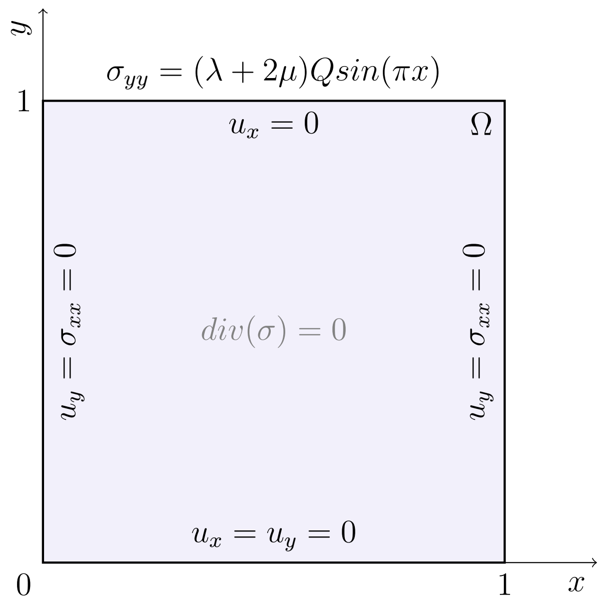

Figure 4: Boundary conditions for the problem defined in [16]

[1]https://github.com/sciann/sciann-applications

The body forces are chosen on purpose so that there can be an analytical solution: [16]

$$\begin{aligned} f_x &= -\lambda\left(-\pi Q y^3 \cos(\pi x) + 4\pi^2 \sin(\pi y)\cos(2\pi x)\right) \\ &\quad - \mu\left(-\pi Q y^3 \cos(\pi x) + \pi^2 \sin(\pi y)\cos(2\pi x)\right) - 8\pi^2 \mu \sin(\pi y)\cos(2\pi x) \\ f_y &= \lambda\left(-3Qy^2 \sin(\pi x) + 2\pi^2 \sin(2\pi x)\cos(\pi y)\right) \\ &\quad + \mu\left(\frac{\pi^2 Q y^4 \sin(\pi x)}{16} - 6Qy^2 \sin(\pi x) + 2\pi^2 \sin(2\pi x)\cos(\pi y)\right) \end{aligned} \tag{16}$$

The corresponding analytical solution is then given by:

$$\begin{aligned} u_x &= cos(2\pi x) \times sin(\pi y) \\ u_y &= sin(\pi x) \times \frac{Qy^4}{4} \end{aligned} \tag{17}$$

For the sake of illustration, the displacement field solution is plotted in figure 5 (all fields are plotted in appendix A.1)

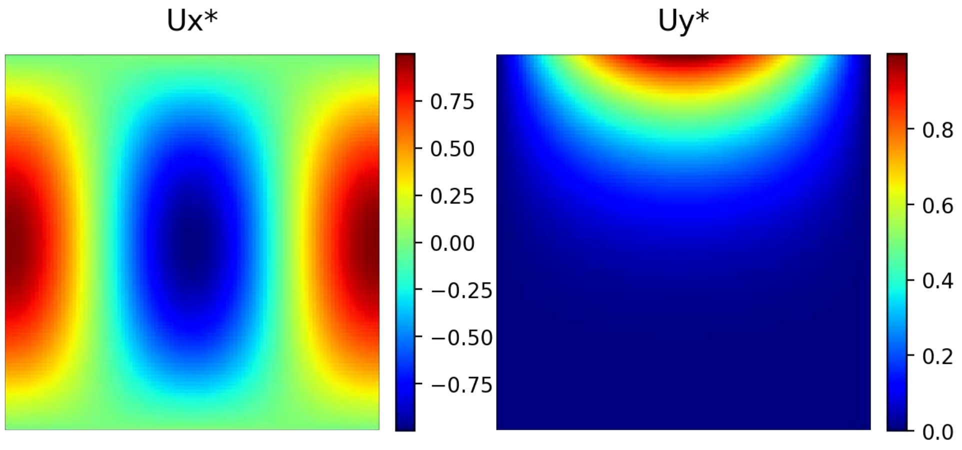

Figure 5: Displacement solution of the problem

The resolution of this analytical problem using SymPy [19] symbolic math python library is available on GitHub[2].

## Implementation

The implementation of the problem by Haghighat et al. [16] is based on SciANN [20] library developed by the authors. They use a PFNN (parallel feedforward neural network) architecture with a different neural network for each component of the tree fields ($u$, $\epsilon$, and $\sigma$). It is important to note that this implementation uses ground truth data of all the fields to enforce the boundary conditions. This is a stronger constraint than the one defined in the BVP. More details on the implementation can be found in the original paper.

[2]`https://github.com/bonneted/pinn-cm`

## Results

To assess the reliability of the implementation, we run the code from [16] 20 times in order to get statistically significant results. A Weight and Biases report of these 20 runs is available on the following link: `https://api.wandb.ai/links/damien-bonnet/73x2h12z`. The loss history of all the runs is plotted in figure 6.

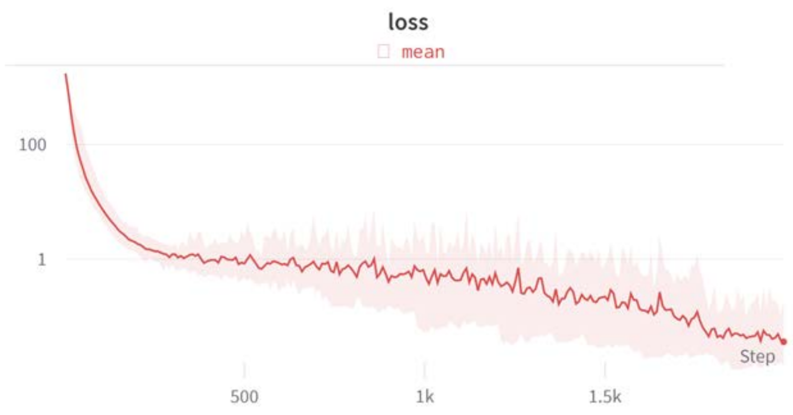

Figure 6: Loss history (minimum, mean, and max) of 20 runs of the SciANN implementation

A plateau is reached at the end of the training with no significant improvement. The histogram of the losses at the end of the training is plotted in figure 7.

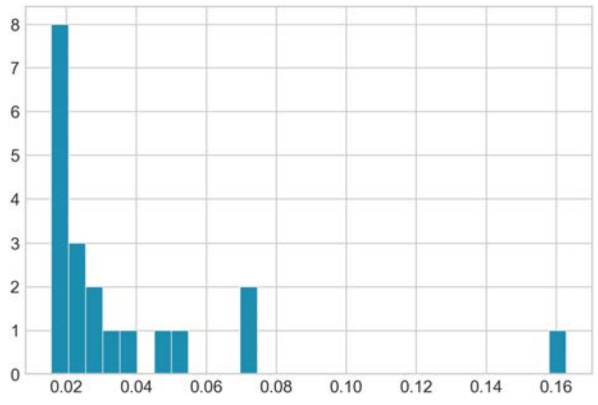

Figure 7: Histogram of the losses at the end of the training

The median run, in terms of final loss, is compared to the analytical solution in figure 8.(worst and best runs are also plotted in appendix A.2).

It is difficult to have a global view of the residuals as all the fields are outputs of the neural network in this approach and should thus be taken into account. That is why we focus on the displacement field. The residual for the median run is around $1 \times 10^{-2}$ on average and can be as high as $2 \times 10^{-2}$ in some regions. Between different runs, the mean residual varies by about $1 \times 10^{-2}$. These results leave room for improvement, especially when one has in mind that this implementation uses ground truth data of all the fields on the boundary.

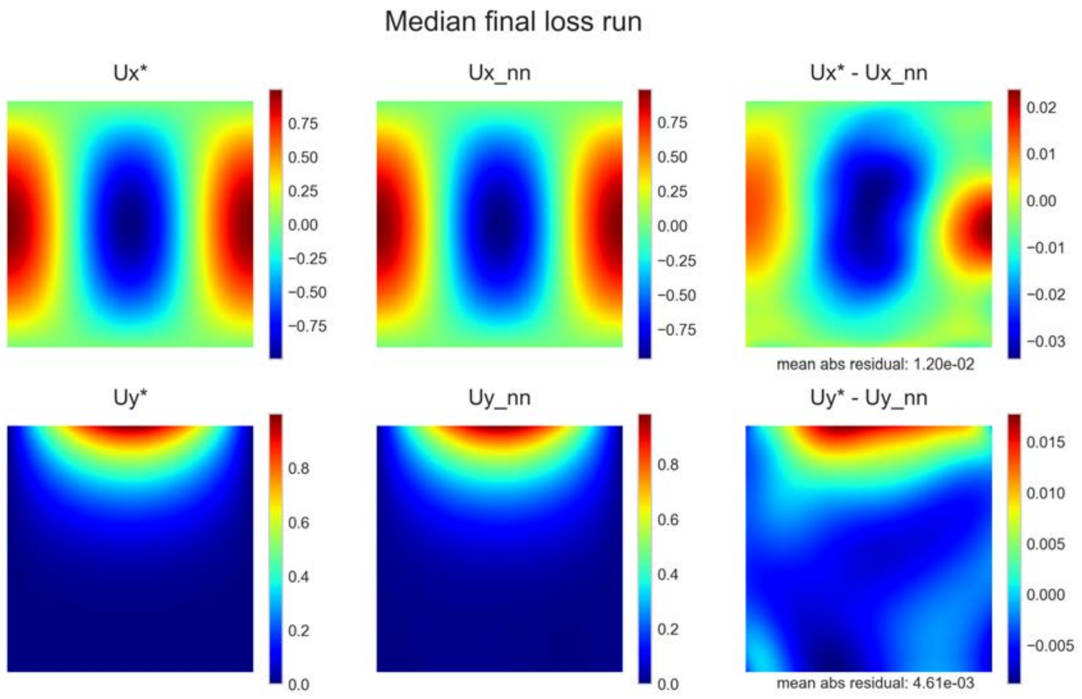

Figure 8: Displacement field approximation of the median run compared to the analytical solution

## 3.3 Enhancing the reliability of PINNs through hyperparameter optimization

Our implementation of the problem has been developed using the DeepXDE library [17], and the code is available on GitHub[3].

Contrary to the SciANN implementation, we want to use only the information from the BVP and no other ground truth data. The BVP is indeed well-defined, meaning that the information it provides is sufficient to find the unique solution to the problem. We will first address a simplified version of the problem, with only displacement boundary conditions, to have a baseline for the mixed boundary conditions problem.

### Simplified BVP (only displacement BC)

The equivalent BVP with only displacement boundary conditions is plotted in figure 9. Having only displacement boundary conditions allows us to strictly enforce all the boundary conditions. It is important to note that this implementation is still more restrictive than the SciANN one, as the latter uses ground truth data of all the fields on the boundary.

For the implementation, we use a simple feedforward neural network with 4 hidden layers of 50 neurons each. The neural network output is the displacement field like in the direct implementation represented in figure 3. All the hyperparameters are summarized in table 4.

The loss history of the training is plotted in figure 10.

The Adam optimizer (used for the first 3000 epochs) appears to be stuck at the same level of accuracy than the SciANN implementation. The LBFGS optimizer (used for the next 15000 epochs) is able to reach a lower loss and converge after approximately 11000 epochs. The approximation reached is far more accurate than the SciANN one (see figure 11), as the residual

[3]`https://github.com/bonneted/pinn-cm`

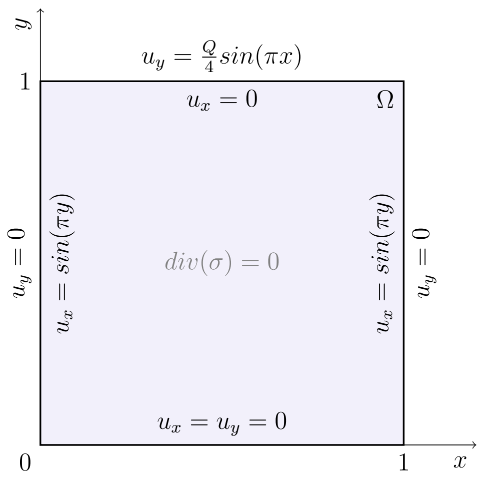

Figure 9: Equivalent BVP with only displacement BC

| Hyperparameter | Value |
|---|---|
| Nb. of hidden layers | 4 |
| Nb. of neurons per layer | 50 |
| Activation function | Tanh |
| Optimizer | Adam then LBFG-S |
| Epochs | 3000 then max 15000 |
| Learning rate | 0.01 |
| Number of samples | 1000 |
| Sampling method | Hammersley |
| BC constraint | hard |

Table 4: Hyperparameters of the simplified BVP

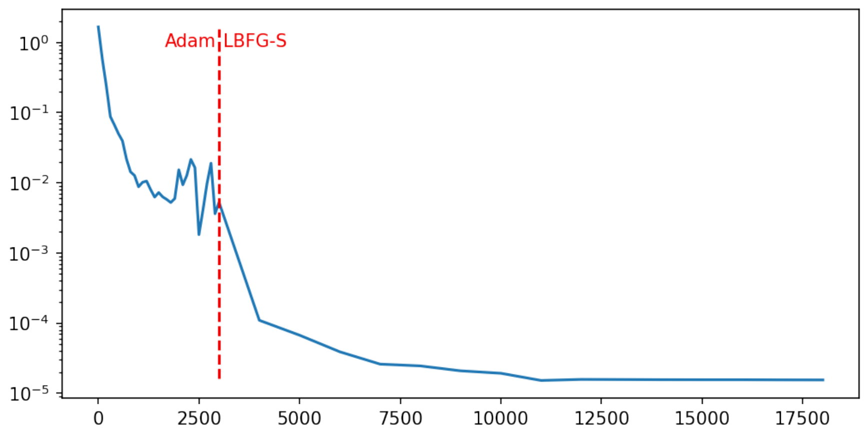

Figure 10: Loss history of the simplified BVP (only displacement BC)

is around $5 \times 10^{-6}$ on average and only goes up to $2 \times 10^{-5}$ in the less accurate regions. The comparison for all fields is plotted in appendix A.3.

These results will be used as a baseline of accuracy for the mixed boundary conditions problem in the next section.

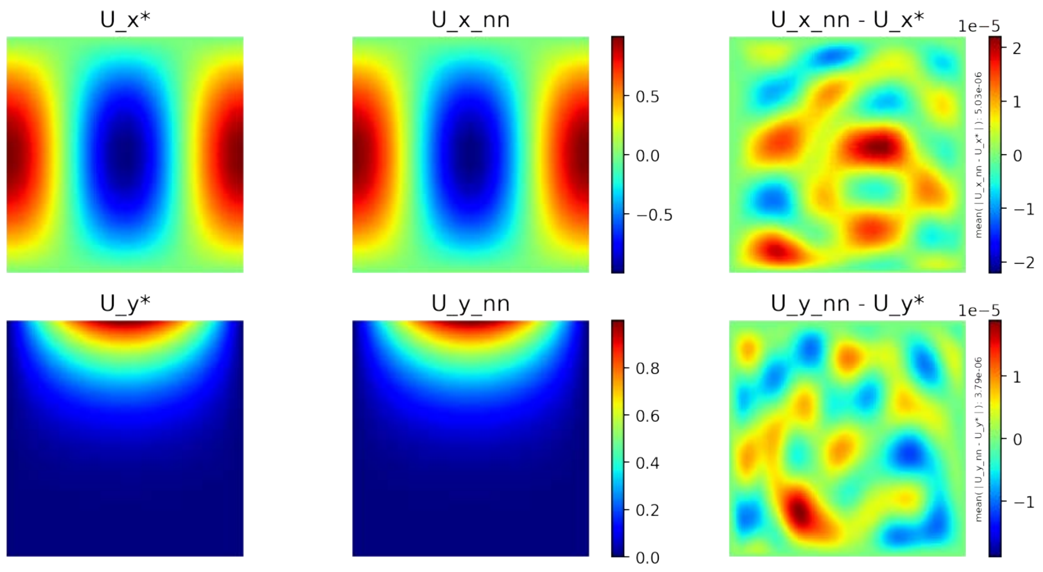

Figure 11: Displacement field approximation of the simplified BVP compared to the analytical solution

### Mixed BVP (displacement and stress BC)

We use the same hyperparameters to solve the mixed BVP with both direct and parallel implementation. The accuracy metric history of the training is plotted in figure 12.

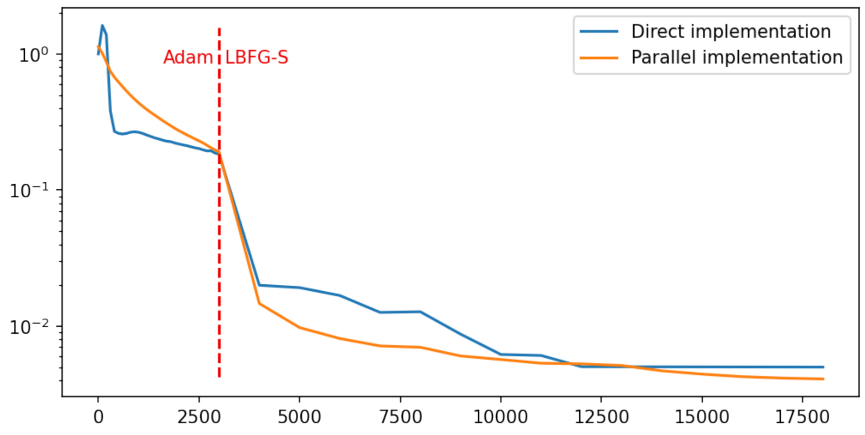

Figure 12: Accuracy metric history of mixed BC problem (for both direct and parallel implementation)

The final loss reached is lower than the SciANN implementation but still three orders of magnitude above the baseline from the simplified BVP. Regarding the displacement approximation, the residual is around $2 \times 10^{-3}$ for both implementations, which is already one order of magnitude better than the SciANN implementation. (displacement fields are plotted in appendix A.4). We investigate in the next section how to improve these results through hyperparameter optimization (HPO).

### Hyperparameter optimization of the problem

To limit the number of experiments, we limit our study to the neural network architecture, the implementation (direct or parallel), and the number of samples used for the training (listed in table 5). The other hyperparameters are kept constant, as they were found to be optimal for the simplified BVP problem.

| Hyperparameter | Possible values |
|---|---|
| Implementation | direct, parallel |
| Nb. of hidden layers | 4, 5, 6 |
| Nb. of neurons per layer | 50, 75, 100 |
| Activation function | Sigmoid, Tanh, ReLU |
| Number of samples | 2500, 10000 |

Table 5: Hyperparameters considered for the HPO

We use the Weights and Biases [21] platform and its hyperparameter optimization (HPO) module Sweeps to track the experiments and visualize the results. There are $2 \times 3 \times 3 \times 3 \times 2 = 108$ experiments to run, which took us a few hours to run (20 minutes per experiment, 10 runs in parallel). The results are available at `https://api.wandb.ai/links/damien-bonnet/m6nfcoy8`. The histogram of the final accuracy metric for each run is plotted in figure 13. The best run improves the accuracy by approximately 40% compared to the first run of the previous section. The main results of this study concern rather the influence of the parameters on the accuracy. Due to the stochasticity of the training, the results must be interpreted as general trends rather than exact values.

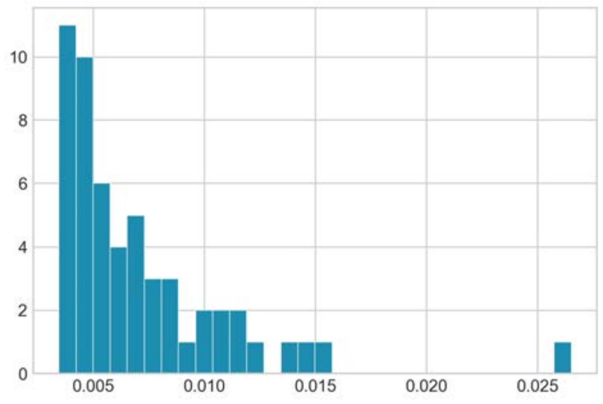

Figure 13: Histogram of the final accuracy metric for each run of the HPO

## Activation function

The first thing to note is that approximately 40% failed before the end, the algorithm returning a NaN loss. After investigation, we found that the ReLU activation function was a major cause of this issue as all runs with this activation function failed when reaching the 3000 epochs (switch to LBFG-S optimizer). The non-regularity of the ReLU function may be pathological for the LBFG-S optimizer. Therefore, we remove this activation function from the HPO, which reduces the failure rate to 15%. Otherwise, Tanh outperforms Sigmoid on average in terms of accuracy, which confirms that Tanh is a better choice for PINNs.

### Network size

The results of the HPO show that the network width and depth have opposite effects on the accuracy. The number of layers is correlated with a worse accuracy, while the number of neurons per layer is correlated with a better accuracy. This encourages the use of wide and shallow networks.

### Implementation (direct or parallel)

The parallel implementation is more accurate than the direct one on average, the history of the accuracy metric grouped by implementations is plotted in figure 14. This result indicates that having a soft constraint on the material law allows better approximations than a soft constraint on the stress boundary condition, which seems coherent from a physical point of view. Indeed, whereas a boundary condition can be known exactly (e.g. zero stress on a free surface), the material law is empirical, and therefore non-exact, which makes it more suitable for a soft constraint. Additionally, the parallel implementation offers a seamless generalization to other material laws, simply by changing the material law loss function.

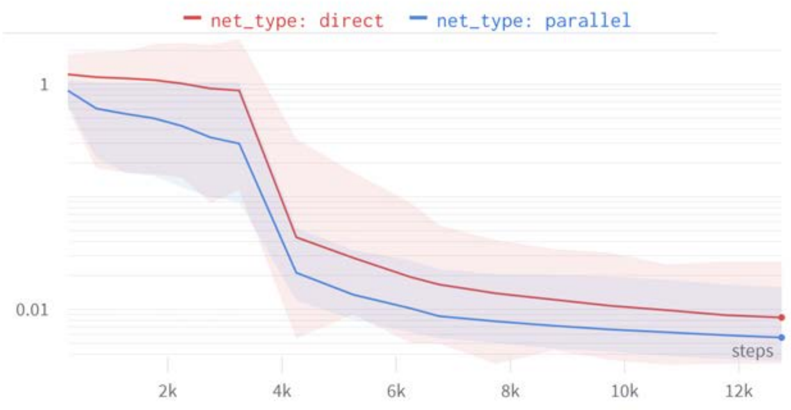

Figure 14: Accuracy metric history of the HPO grouped by implementations (min, mean, and max)

## 4 CONCLUSION

In this paper, after introducing the PINN framework, we proposed two implementations (direct and parallel) of PINNs to solve boundary value problems in continuum mechanics. Focusing on a linear elastic problem, we assess the accuracy of an existing PINN implementation [16] before improving it through hyperparameter optimization. Even though the improvement achieved is limited (40% improvement), we were able to identify the main parameters that influence the accuracy of the PINN approximation. Concerning the network architecture, we found that wide and shallow networks with a Tanh activation function perform better. Concerning the implementation, we found that the parallel implementation is more accurate than the direct one, which is consistent with the fact that the material law is more suitable for a soft constraint than the boundary conditions. Nevertheless, the same network architecture performed 2 orders of magnitude better on a simplified equivalent problem, proving that the training error is still limiting the accuracy. To avoid getting stuck in local minima, different further improvements could be investigated, such as the use of adaptive sampling methods, learning rate schedules, or more complex network architectures. These results are promising and encourage the use of parallel implementation of PINNs for inverse quantification of material properties using DIC measurements.

## ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie project GREYDIENT – Grant Agreement n°955393.

## REFERENCES

[1] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, pp. 422–440, June 2021.

[2] C. M. Hamel, K. N. Long, and S. L. B. Kramer, "Calibrating constitutive models with full-field data via physics informed neural networks," Mar. 2022.

[3] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, pp. 1026–1030, Feb. 2020.

[4] M. Rossi, P. Lava, F. Pierron, D. Debruyne, and M. Sasso, "Effect of DIC Spatial Resolution, Noise and Interpolation Error on Identification Results with the VFM: Effect of DIC Spatial Resolution, Noise and Interpolation on VFM Identification," *Strain*, vol. 51, pp. 206–222, June 2015.

[5] J. Martins, A. Andrade-Campos, and S. Thuillier, "Comparison of inverse identification strategies for constitutive mechanical models using full-field measurements," *International Journal of Mechanical Sciences*, vol. 145, pp. 330–345, Sept. 2018.

[6] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec. 1943.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, Dec. 1989.

[9] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, Jan. 1993.

[10] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014.

[11] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, pp. 503–528, Aug. 1989.

[12] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, Sept. 1998.

[13] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial

differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019.

[14] N. Sukumar and A. Srivastava, "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114333, Feb. 2022.

[15] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes, "A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 379, p. 113741, June 2021.

[16] E. Haghighat, M. Raissi, A. Moure, H. Gomez, and R. Juanes, "A deep learning framework for solution and discovery in solid mechanics," May 2020.

[17] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, pp. 208–228, Jan. 2021.

[18] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 403, p. 115671, Jan. 2023.

[19] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, Am. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "SymPy: Symbolic computing in Python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017.

[20] E. Haghighat and R. Juanes, "SciANN: A Keras/Tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 373, p. 113552, Jan. 2021.

[21] L. Biewald, "Experiment tracking with weights and biases," 2020. Software available from wandb.com.

## A APPENDIX

### A.1 Analytical solution of the problem studied in section 3.2

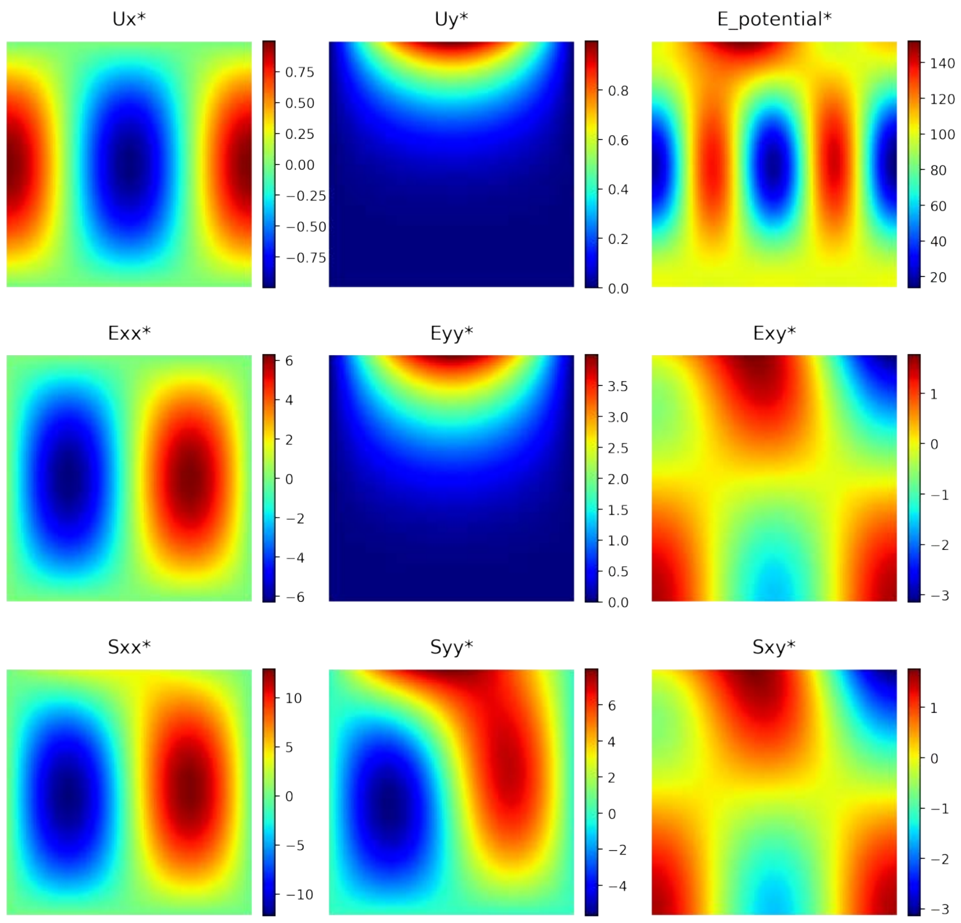

Figure 15: Analytical solution of the problem studied in section 3.2

## A.2 Results of the SciANN implementation

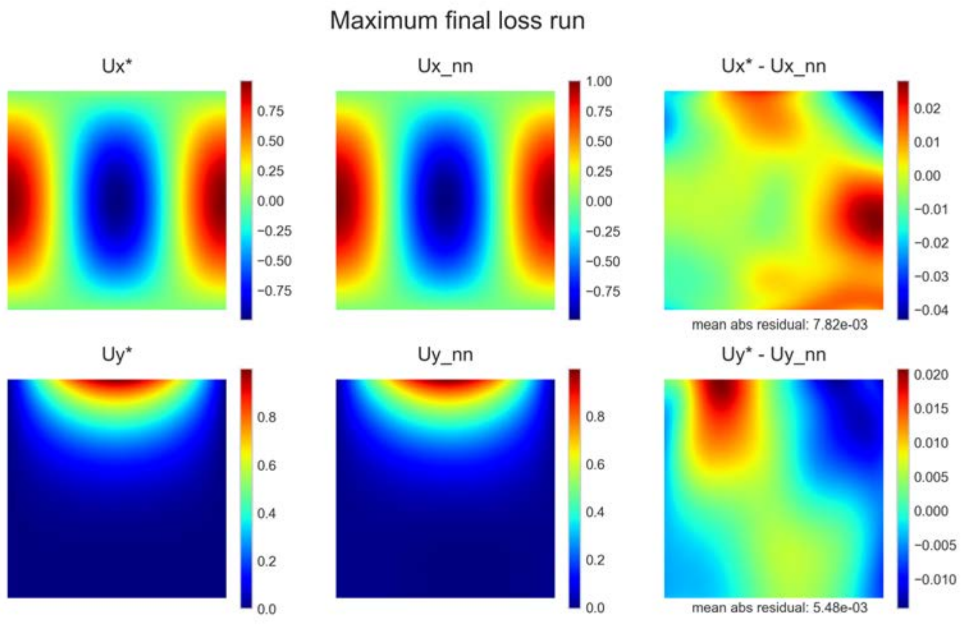

Figure 16: Displacement field approximation of the worst run compared to the analytical solution

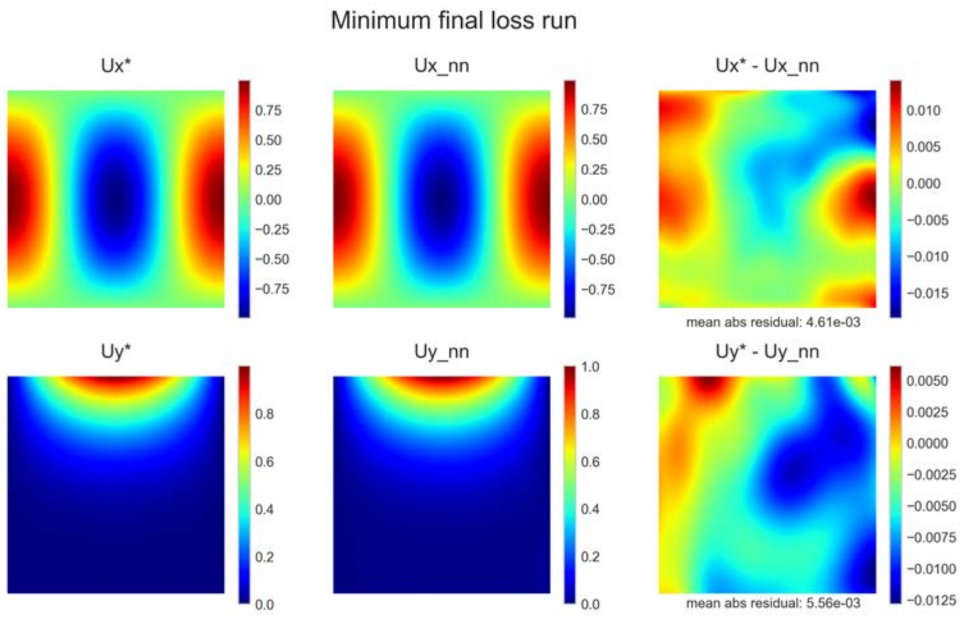

Figure 17: Displacement field approximation of the best run compared to the analytical solution

## A.3 Results of the simplified BVP (only displacement BC)

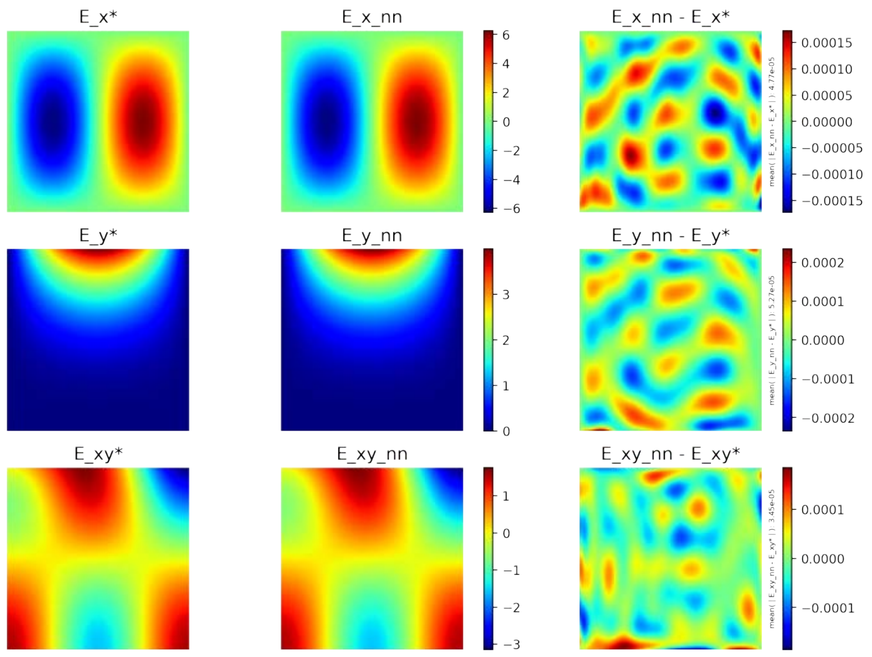

Figure 18: Strain field approximation of the simplified BVP (only displacement BC) compared to the analytical solution

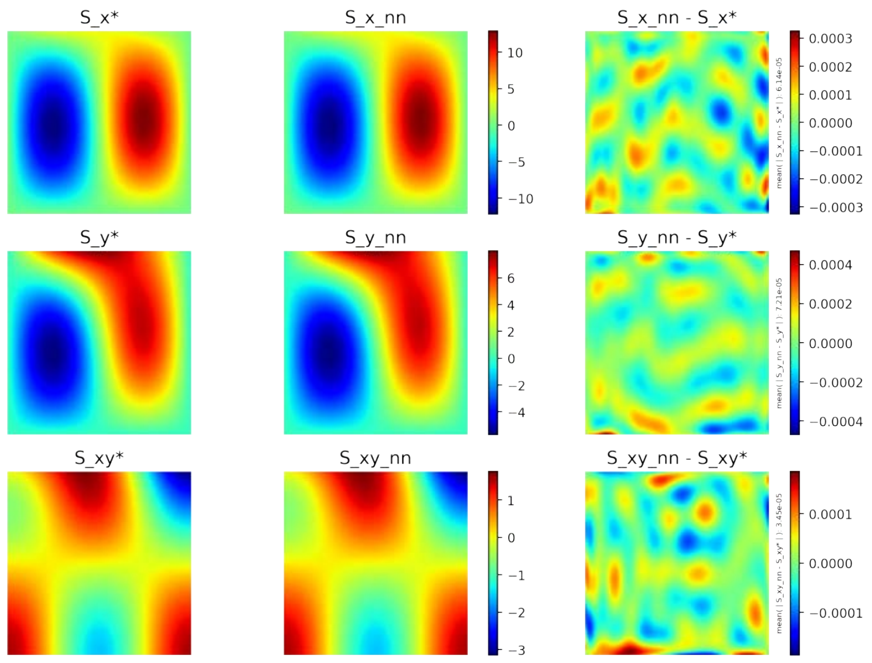

Figure 19: Stress field approximation of the simplified BVP (only displacement BC) compared to the analytical solution

## A.4 Results of the mixed BVP before HPO

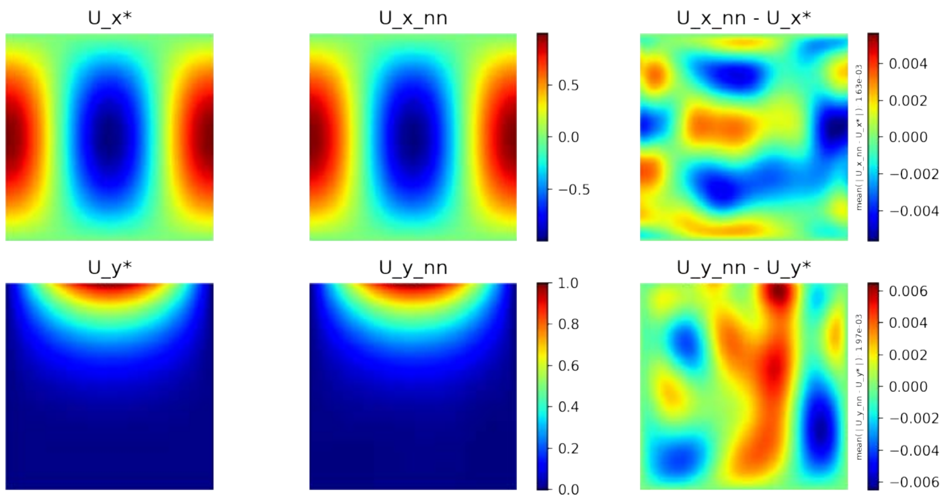

Figure 20: Displacement field approximation of the mixed BVP (direct implementation) compared to the analytical solution

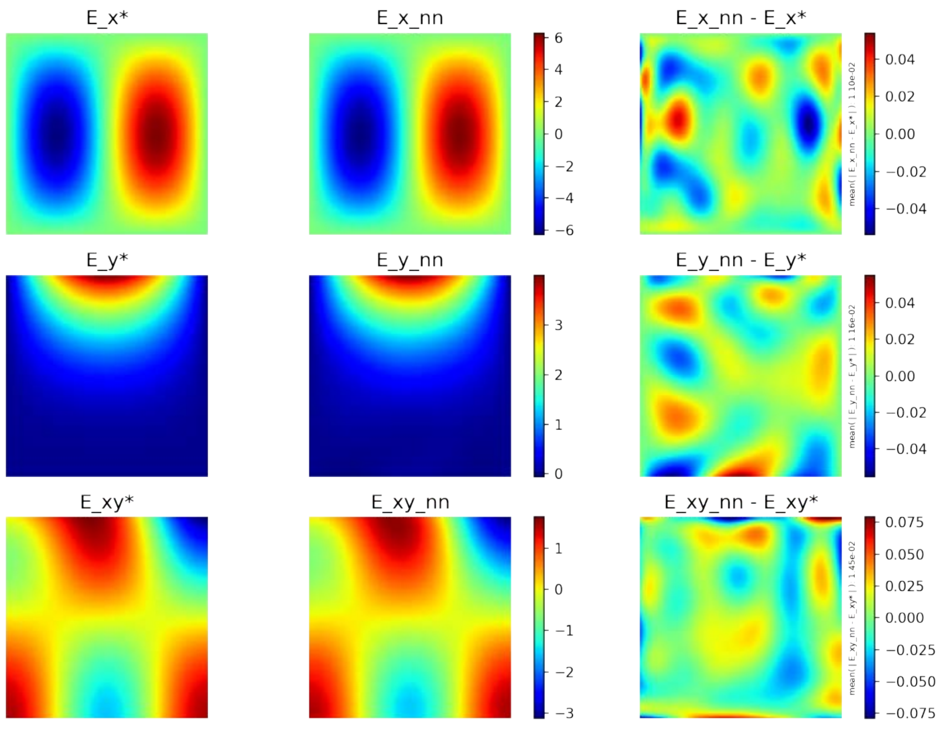

Figure 21: Strain field approximation of the mixed BVP (direct implementation) compared to the analytical solution

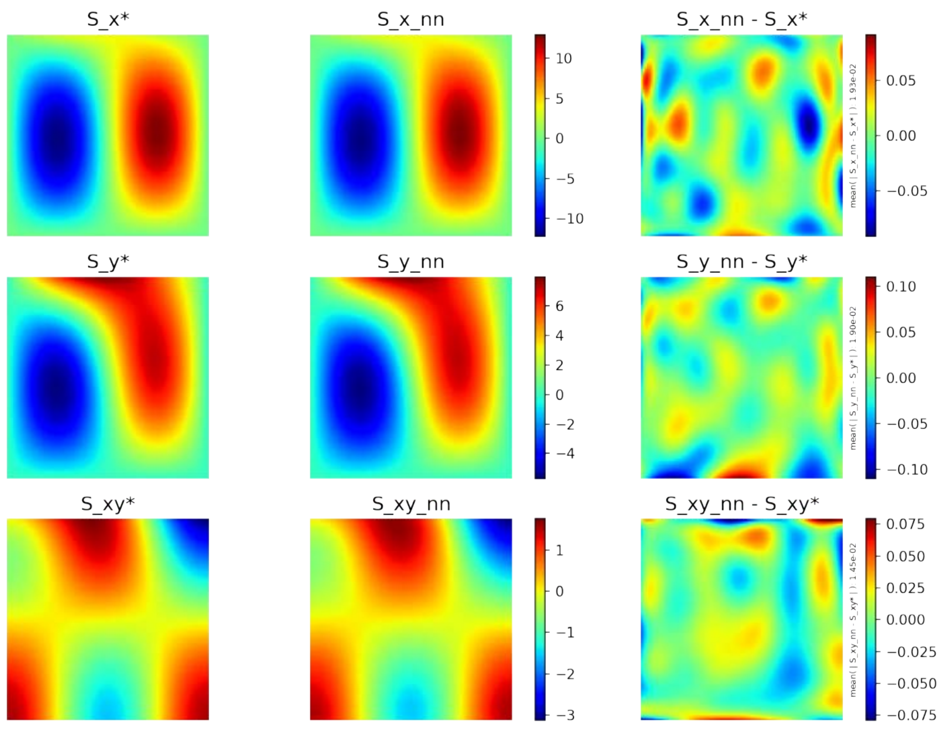

Figure 22: Stress field approximation of the mixed BVP (displacement and stress BC) compared to the analytical solution