POLITECNICO DI TORINO

Master's Degree in Engineering and Management

Master's Degree Thesis

Physics-informed machine learning for manufacturing applications

Supervisors

Prof. Giulia BRUNO

Emiliano TRAINI

Gabriel ANTAL

Candidate

Paolo FINUCCI

APRIL 2025

Summary

Physics-Informed Machine Learning (PIML) has emerged as an innovative approach that integrates previous knowledge from physics into machine learning models, improving their ability to solve engineering problems. Within the PIML framework, Physics-Informed Neural Networks (PINNs) have gained importance due to their ability to enforce physical laws as constraints directly during the training process, reducing the need for large and labeled datasets.

This thesis begins with a theoretical exploration of PIML, distinguishing PINNs as a specific class of physics-informed machine learning models. We then analyze the core principles and key challenges of PINNs, including simple applications of the framework to illustrate its foundational concepts. Building on this theoretical background, we applied the PINN framework to engineering and manufacturing scenarios, starting with a 2D linear elasticity model, where a PINN is used to approximate stress and displacement fields subject to physical constraints. The study is then extended to a 2D transient heat diffusion problem and after reviewing the theoretical background of Resistance Spot Welding (RSW), we developed a parametric equation to estimate the nugget growth and implemented it with an informed machine learning approach.

A comparative analysis with traditional numerical methods, specifically the Finite Element Method (FEM), is done to validate the predictions of the PINN models for the simulated scenarios.

Acknowledgements

Dedico questo lavoro alla mia famiglia, per avermi sostenuto durante questo percorso e ai miei amici, per averlo reso ancor più significativo.

Table of Contents

List of Tables VII

List of Figures VIII

1 Introduction 1

2 Theoretical Foundations of Physics-Informed Machine Learning 3

2.1 Theoretical Background . . . 3

2.1.1 Foundations of Machine Learning . . . 3

2.1.2 Physics-Informed Machine Learning (PIML) . . . 6

2.2 Introduction to Physics-informed Neural Networks (PINN) . . . 8

2.2.1 Foundations of Artificial Neural Network . . . 8

2.2.2 Limits of Classical ANN Models . . . 12

2.2.3 Foundations of Differential Equations . . . 12

2.2.4 Theoretical background of PINNs . . . 15

2.3 PINNs applications . . . 16

2.3.1 1D Burgers Equation . . . 17

2.3.2 1D Heat equation . . . 19

2.3.3 2D Transient Heat Equation (DeepXDE) . . . 20

2.4 Challenges and Limitations . . . 24

3 Application of a PINN to model a linear elasticity problem 26

3.1 Governing Equations . . . 27

3.2 Python Implementation . . . 29

4 Approaching multi-physics models with Resistance Spot Welding 42
4.1 Theoretical background 42
4.1.1 Governing Model 43
4.1.2 Challenges on RSW modelilng with a traditional PINN . . . 46
4.2 PINN for Modeling 2D Heat Diffusion with a Gaussian Distributed Heat Source 48
4.2.1 Physical Model 48
4.2.2 DeepXDE implementation 50
4.2.3 Results 54
4.3 Parametrization with informed ML to estimate nugget growth . . . 57
4.3.1 ODE Model 58

5 Conclusions and future work 60
5.1 Benefits and Challenges of PINNs approaches in industrial context . 60
5.2 When to Use PINNs vs. FEM: Considerations and Analytical Solutions 61

Bibliography 62

List of Tables

2.1 Advantages and Challenges of PINNs 16

List of Figures

2.1 This is a sample caption for the figure. 5
2.2 Structure of an artificial neuron 9
2.3 Simple ANN structure 10
2.4 PINN accuracy compared to the exact solution. 17
2.5 PINN accuracy compared to the exact solution. 18
2.6 Error contourn plot for 1D Burgers Equation PINN 19
2.7 Heat map for the 1D rod 21
2.8 Decreasing loss value and heat map for a 2D heat equation applied on a 1x1 square 23
2.9 Heat map for t=0.55 seconds, with Zero Neumann BC and Robin BC. 24

3.1 Code snippet of the implementation of governing equations. 30
3.2 Code snippet for the oscillating side modeling. 30
3.3 Implementation of all the boundary and initial conditions. 31
3.4 FEM data as boundary conditions 31
3.5 PINN data 32
3.6 Model definition, configuration and training 33
3.7 Model prediction, input is a 2D array with time and space coordinates. 33
3.8 PINN prediction evaluated at 0,2 s. 35
3.9 Exact FEM solution at 0.2 s. 36
3.10 Contour graph for the PINN error evaluated against the FEM solution. 37
3.11 Line plots for u_x and u_y 40
3.12 Line plot for σ_{xx} 41

4.1 Governing equations of the model. 50
4.2 Gaussian distribution for heat profile. 51
4.3 52
4.4 Enter Caption 53
4.5 Model definition and configuration 53
4.6 Heat map based on numerical simulation data. 54
4.7 PINN prediction. 55
4.8 Error map for the PINN solution compared against FEM results. . . 56
4.9 Line plot at spatial coordinates (0.5;0.25). 57

Chapter 1

Introduction

Machine learning and physics can be used together in a new approach called Physics-Informed Machine Learning (PIML). Differently from classical machine learning, which relies mostly on big dataset for the training, PIML includes physics rules into the learning process. This makes the models more accurate, especially for engineering problems where getting experimental data can be difficult. One of the most important methods in PIML is Physics-Informed Neural Networks (PINNs). These models use deep learning while following physical laws, like differential equations. Traditional methods, such as the Finite Element Method (FEM), have been used for many years to solve physics-based problems, but they can be slow and require a lot of computing power. PINNs offer a different approach in order to solve these problems because it learns also from equations instead of just data, making them useful for applications like heat transfer, material science and fluid flow. This thesis starts, in chapter 2, by explaining the theory behind PIML and PINNs to better understand the basics of artificial neural networks and their limits, as well as the key type of equations, boundary conditions and initial conditions that PINNs use. To better understand these concepts, initially the thesis looks at simple physics problems, like the 1D Burgers equation, 1D heat equation, and 2D transient heat equation. These examples show how PINNs can approximate solutions while following physics laws. After covering the basics, in chapter 3 the thesis starts to focus on real engineering problems. The first studied example is

a PINN application on a 2D linear elasticity model, where the model predicts stress and displacement in a cantilever subject to oscillation on the free side. The predictions of the PINN are validated against FEM simulation results.

Then, in chapter 4, the thesis approaches the study of the physical model of resistance spot welding to assess the feasibility of such implementation using physics informed machine learning methods. In this case, after developing a parametrized governing model for the welded nugget diameter, we developed a hybrid Physics informed machine learning model able to approximate the equation parameters based on a small dataset and then compute the nugget diameter. In addition to this model the thesis presents an application of a PINN for an heat diffusion problem where a metal plate is heated at the center by an heat source modeled as joule heating.

Finally, in chapter 5, the thesis compares PINNs with traditional methods like FEM, highlighting pros and cons for both applications. This work helps to connect physics and machine learning, showing that PINNs are not just useful for basic problems but can also solve real-world engineering challenges. By using both machine learning and physics, PINNs offer a new and more efficient way to solve complex engineering problems.

Chapter 2

Theoretical Foundations of Physics-Informed Machine Learning

Physics Informed Machine Learning is a widely researched field at the time this thesis is being redacted and it focuses on integrating machine learning techniques with physics models with the objective of improving ML generalization capabilities, reducing the dimension of the dataset requirements for the training and overall to improve the model prediction accuracy.

2.1 Theoretical Background

2.1.1 Foundations of Machine Learning

Machine learning (ML) is a subgroup of artificial intelligence (AI) that focuses on developing algorithms and statistical models that allows computers to perform specific tasks without explicit instructions. In fact, these systems learn patterns and make decisions based on available datasets, making machine learning a very useful tool for solving complex problems across a wide range of fields. In the modern times, ML in integrated in services used on a daily basis by most of the population, for

example, in recent years, Amazon started recommending products a costumer may find useful, predicting their tastes according to their habits and interests. Also face recognition is nowdays implemented by most of the smartphones, allowing to sort photos by the people showing in them. Another example of everyday application is represented by e-mail services: they integrate ML in the automatic spam filtering, by finding common patterns in spam email, the algorithm is able to automatically detect unwanted mail with high accuracy. Machine Learning, at its core, relies on three core principles: data, algorithms and models; these elements form the skeleton of each model.

- **Data**: Database containing training sets. Usually, they are linked to each other and the ML model is built in order to predict one of them based on the apparent relationship with the others (example: estimate houses price based on the number of bedrooms, floor and neighborhood).
- **Algorithms**: It is the sets of rules that guide the learning process.
- **Models**: It is the output of the learning process, incorporating the patterns and relationships that lies within the data used for training.

The development of a working ML model involves different steps. In fact, only feeding data into an algorithm doesn't allow to capture all the relevant relationships needed for an accurate and reliable modeling of a particular process; in particular the process can be summarized into the following stages:

- **Training**: It is the process in which the model learns to make viable predictions, using a subset of the data to adjust the model's parameters to minimize prediction error.
- **Tuning**: It's the process of optimization of the hyperparameters of the model in order to improve performance. In fact, hyperparameters are not learned during the training step but are directly set during the code implementation.
- **Validation**: As defined by [1], in this step the fitted model is applied to a different dataset (validation set) not used in the training phase, providing an

unbiased evaluation of the model fit on the training dataset while tuning the model's hyperparameters.

- **Testing**: It is the final evaluation of the final tuned model when comparing it to different final models.

While the final goal of a ML model is to obtain generalization, i.e. performing well on new, unseen data, capturing all the underlying patterns, there are some possible inefficiencies that can affect the reliability of the model, as shown in fig. 2.1:

- **Overfitting**: It can happen when the model is too complex, forcing the model to capture the underlying patterns but also the noise that lies in the training data, building a model that performs very well on training sets but poorly when applied to new unseen data.
- **Underfitting**: It can happen when the model is too simple (for example, when applying a linear model to a very complex problem). In this case, the model will not capture the underlying patterns and will perform poorly both on the training data and on new datasets.

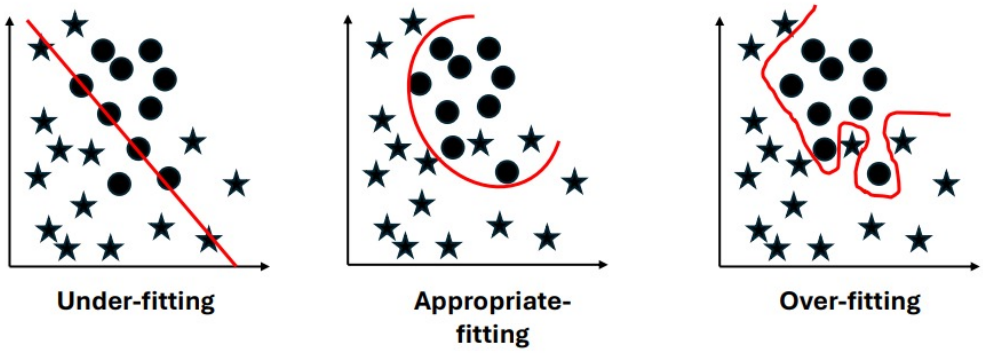

Figure 2.1: This is a sample caption for the figure.

ML models, based on the types of data used during training (labeled vs unlabeled) and training methods, can be grouped into three main categories:

- **Supervised Learning**: The ML model learns from a labeled dataset, where each input has a corresponding output. In this way, the model learns to associate inputs with outputs and can make predictions on new data according to the input-output map. Some of the most used algorithms are linear regression, support vector machines, decision trees, and neural networks.
- **Unsupervised Learning**: The ML model is trained on unlabeled datasets, grouping the data based on input similarities and patterns.
- **Semi-Supervised Learning**: This is a link between supervised and unsupervised learning, where the dataset used for training is composed of both labeled and unlabeled data.
- **Reinforcement Learning**: It is a feedback-based process where the AI interacts with its surroundings using a trial-and-error mechanism. The AI is rewarded for good actions and penalized for mistakes, reinforcing good behavior or predictions, with the aim of maximizing its "score".

2.1.2 Physics-Informed Machine Learning (PIML)

There is still not an universally accepted and clear definition for physics informed machine learning, the definition taken in consideration in this work thesis is the one stated by Rueden et al. in “Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems”, is that the essence of informed machine learning consists in the integration of prior knowledge (for example a mathematical formula) - characterized by its representation methods - into the machine learning pipeline. In the following section I’ll investigate various methods to build a physics-informed or physics-guided machine learning model, embedding previous knowledge in the form of mathematical relationship of various complexities to guide the model predictions. https://dl.acm.org/doi/pdf/10.1145/3514228 (Willard)

As the field of machine learning evolves, effectively integrating domain-specific knowledge is essential to develop effective models. "Knowledge-guided Machine Learning: Current Trends and Future Prospects" by Karpatne et al. distinguishes

between three main aspects to be considered when working with a Physics-Informed Machine Learning (PIML) framework:

- **Type of Scientific Knowledge**: The first consideration is the type of scientific knowledge available; it can take various forms, for example, first principle equations, domain-specific laws, rules, or heuristics. It is also possible to encounter systems for which the knowledge is only partial or imperfect because the process is not fully understood.

  In order to incorporate such knowledge, it is possible to employ the framework of physics-guided neural networks (PGNNs), initially developed for modeling lake temperatures using monotonic constraints. This has increased research efforts into integrating various forms of scientific knowledge, such as eigenvalue equations, energy balance, and scaling laws. For instance, physics-guided recurrent neural networks (PGRNNs) have been used to predict lake temperatures while accounting for energy conservation over time.

  When the process is fully understood and described by exact equations, for example with Partial Differential Equations (PDEs), it is viable to employ frameworks such as Physics-Informed Neural Networks (PINNs), which will be discussed in depth in the next section.

- **Form of Physics-ML Integration**: ML models can be differentiated based on how scientific knowledge is integrated with machine learning, ranging from ML-centric to process-centric approaches. The primary role of ML methods is to map inputs to response variables, with scientific knowledge guiding the algorithms towards scientifically realistic solutions. In process-centric approaches, ML can be used to identify or calibrate parameters, improving the match between model outputs and observations. The two forms can also be combined in hybrid modeling, creating approaches where the final prediction is the product of both integration methods.

- **Method for Incorporating Scientific Knowledge in ML**: Karpatne et al., in "Knowledge-guided Machine Learning: Current Trends and Future Prospects," differentiate between three methods to incorporate scientific

knowledge in ML models, all of which can be used together:

1. **Knowledge-based Learning**: The learning algorithm is changed to select models that are more compatible with scientific equations. This approach is employed in PINNs.
2. **Knowledge-guided Architecture**: The neural network structure is selected beforehand according to the characteristics of the physical model in analysis (used in Physics-guided architecture, PGA). This approach allows the knowledge to be "implicitly embedded" into the ML framework.
3. **Knowledge-guided Pretraining**: Neural network parameters are initialized with specific weights that are derived from previous knowledge.

2.2 Introduction to Physics-informed Neural Networks (PINN)

2.2.1 Foundations of Artificial Neural Network

During the conducted research the interest shifted towards PINNs for their good integration between strong physical models, low dataset requirements for the training and good accuracy. In the following section it will be presented a short introduction on classical neural networks methods before diving into Physics-Informed Neural Networks theoretical background and applications.

Artificial Neural Networks (ANN) are computational models, introduced by mathematician and neurophysiologist Warren McCulloch in 1943, inspired by the human brain that can learn and extrapolate patterns from data.

Overall, a generic ANN is a supervised learning algorithm, composed of artificial neurons, arranged in a variable number of layers and exchanging signals (numbers) between each other (simulating synapses in biological brains).

The neuron is the fundamental computational unit of a neural network. Each neuron receives an input, in the form of $(X_1, X_2, \ldots, X_n)$, composed of a set of values, each associated with a parameter called weight $(w_1, w_2, \ldots, w_n)$ which represents the relative importance of the corresponding input in the neuron function.

The summation function aggregates the weighted inputs, mathematically expressed as:

$$Y_{\text{pred}} = \sum_{i=1}^{n} w_i \cdot X_i + b$$

where b represents the bias, an additional parameter that helps in shifting the activation function to better fit the data. After the summation takes place, the raw result is passed through an activation function f, which introduces non-linearity into the model, allowing it to learn complex patterns. The appropriate function depends on the specific problem. The most common activation functions are:

- **ReLU**: Outputs zero for negative values and the input itself for positive values, meaning the neuron doesn't activate if the input is zero or less.
- **Sigmoid**: Outputs values between 0 and 1, and only outputs positive signals.
- **Tanh**: Outputs values between -1 and 1, used for tasks where outputs need to be symmetric and centered around zero.

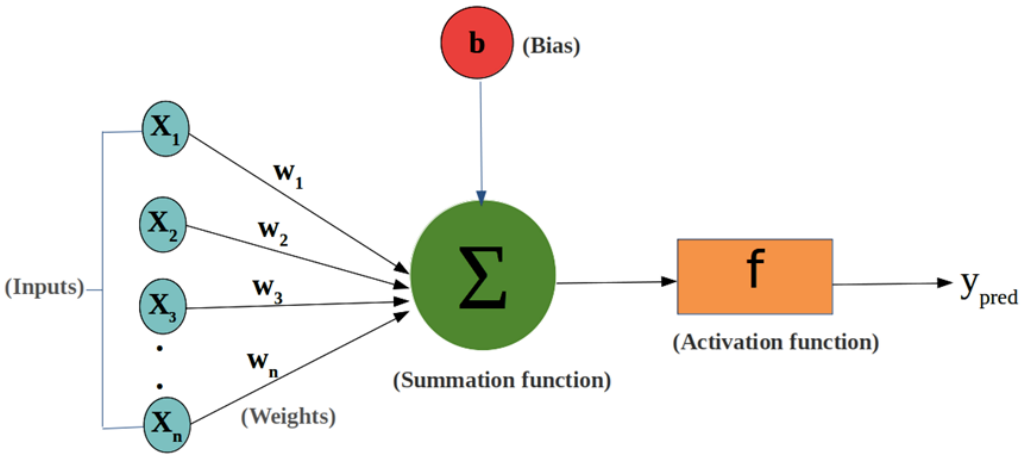

Figure 2.2: Structure of an artificial neuron

The final result is the output of the neuron, denoted as Y_{pred}. This output is

then used as input for the next layer of neurons, and the process repeats until the final output layer.

The simplest form of a neural network is the *feedforward neural network*, where the information flow, as the name suggests, can only go forward through the layers. The number of hidden layers is problem-dependent; usually, increasing the number of layers allows the model to better learn the patterns but also increases the chances of overfitting.

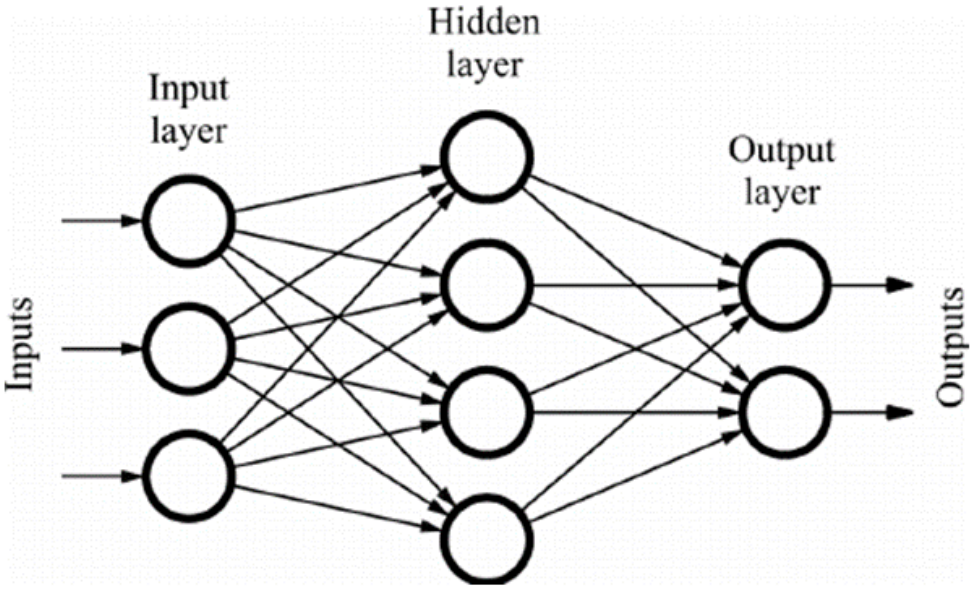

Figure 2.3: Simple ANN structure

The model, during its training, needs to "understand" and differentiate a good prediction from a bad prediction. To fulfill this task, loss functions are implemented, i.e., components that quantify how "wrong" the network's prediction is compared to the target values; the lower the value of the loss function, the better the performance.

During the training, each prediction the network makes is compared to the exact values, obtained by FEM simulations or empirical data. The result is used to adjust the network's weights through a process called *backpropagation*, reducing the loss at each iteration. There are different types of loss functions, tailored to

different machine learning tasks, and the most commonly used are:

- **Mean Squared Error (MSE)**: Used for regression tasks, it calculates the average squared difference between predicted and target values. It penalizes larger errors more. The general formula is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Mean Absolute Error (MAE)**: Used for regression tasks, it calculates the average of the absolute difference between predicted and target values. It is less sensitive to outliers compared to MSE. The general formula is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Cross-Entropy Loss (Log Loss)**: Used for classification tasks, it measures the difference between the predicted probability distribution and the actual distribution, making it perfect for models whose aim is to output probabilities. The general formula is:

$$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

To minimize this loss function, a set of algorithms is employed, each useful for different tasks. The most commonly used are:

- **Gradient Descent**: Iteratively adjusts the weights in the direction of the negative gradient of the loss function, slowly moving towards a minimum.
- **Adam**: Adapts the learning rate for each parameter by maintaining and updating running averages of both the gradient and the squared gradient, making it a good fit for models with a large number of parameters.
- **L-BFGS**: It approximates the second-order derivatives to find the optimal parameters more efficiently. The latter, being computationally very expensive,

is often used as a "refining" optimization technique in the later stages of training to fine-tune the model parameters.

2.2.2 Limits of Classical ANN Models

Despite the impact of NNs in the field of machine learning, for example in image recognition, language processing, and regression analysis, these models display several limitations when applied to problems that require the integration of physical laws.

One of the main limitations of classical NNs is their huge reliance on large amounts of labeled data for training. In fact, they learn patterns from data in a data-driven way: the better the quality of the dataset used for training, the more reliable the output. The natural consequence is that when the data is scarce or noisy, NNs struggle to generalize, providing poor performance on unseen data, occasionally leading to predictions that are inconsistent with known scientific principles if applied to physical models.

Speaking about the latter case, classical NNs cannot be applied to problems governed by partial differential equations (PDEs) because they are not designed to inherently respect the mathematical structure of these equations and often produce solutions that do not satisfy them.

In recent years, a new approach to neural networks has emerged, the so-called Physics-Informed Neural Networks (PINNs). These models are built with the purpose of integrating physical laws, represented by PDEs, directly into the loss function of the networks, with the aim of making the model predictions fit the data and be consistent with scientific principles. The strength of PINNs lies in their ability to produce more accurate and physically consistent models, requiring less data for training and being able to better generalize to new scenarios, offering a powerful tool to solve complex problems where classical NNs used to struggle.

2.2.3 Foundations of Differential Equations

A differential equation is an equation that relates one or more unknown functions and their derivatives. An ordinary differential equation (ODE) involves functions

of a single variable and their derivatives. An example of ODE is $\frac{dx}{dy} = f(x, y)$. A partial differential equation (PDE) is a mathematical equation that involves two or more independent variables, an unknown function (dependent on those variables), and partial derivatives of the unknown function with respect to the independent variables. For example,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

is a first order PDE. The order of an equation is the highest derivative that appears in one of its terms.

Differential equations can also be linear or non-linear:

- **Linear:** when the dependent variable (and its derivatives) appear to the power of one.
- **Non-linear:** when the equation involves nonlinear terms of the dependent variable (or its derivatives); for example, raised to the power of 2 or the presence of a product of derivatives.

Differential equations are fundamental tools in physics because they allow for the modeling of different phenomena. Each can be accurately modeled with ODEs or PDEs depending on their complexity. For example, ordinary differential equations are typically used for systems in a single variable with a finite number of degrees of freedom. Newton's second law ($F = ma$) can be described by an ODE, in fact, being the acceleration the second derivative of position $x(t)$ with respect to time t, it can be expressed in the form:

$$F = m\frac{d^2x}{dt^2}$$

Other processes that involve multiple variables and have an infinite number of degrees of freedom, like wave propagation or heat transfer, are accurately represented by one or more partial differential equations. Their ability to describe how physical variables change over space and over time makes them generally more complex, requiring sophisticated methods to obtain a solution.

PDEs generally have many solutions. This behavior would create inconsistencies

when modeling physical processes, so the set of viable solutions can be narrowed by imposing auxiliary conditions: boundary (BC) and initial conditions (IC).

- **Initial conditions (IC):** specify the physical state at a specific time coordinate, for example at the very beginning of the process. In the heat diffusion process, the IC can specify the initial heat distribution.

- **Boundary conditions (BC):** specify the behavior of the solution at the boundaries of the domain. The main types of BC are:

  - **Dirichlet BC:** specifies the value of the function at the boundary. For example, in a temperature distribution problem, it specifies the temperature at the edges of a metal plate.

  - **Neumann BC:** specifies the value of the derivative of the function normal to the boundary. In heat transfer, it can be used to represent the heat flux at a surface.

  - **Robin (Mixed) BC:** is a combination of Dirichlet and Neumann conditions, specifying a linear combination of the function value and its derivative at the boundary. In a heat transfer problem, it can be used to model the heat transfer due to convection between a solid surface and a fluid in contact with it.

The solution to a differential equation is essential to understanding the behavior of the dynamic system being represented. For ODEs, which involve functions of a single variable, solutions can generally be found using analytical techniques, like separation of variables or integrating factors.

On the other hand, partial differential equations involve more than one independent variable, and one of the obvious consequences is that the difficulty in solving them increases significantly, often preventing analytical methods from being used. In that case, we employ numerical methods, such as the finite difference, finite element, or spectral method, in order to discretize the domain.

2.2.4 Theoretical background of PINNs

One of the research areas in physics in the modern times involves the use of neural networks architectures to model and predict physical processes. Classical neural networks work by approximating, thanks to basic algorithms like linear and polynomial regression, the final output of a process by an extensive training on previously obtained datasets. This approach, fairly accurate in a lot of cases, has a lot of limitations when applied on real, non-laboratory processes, leading to wrong predictions and mistakes. One of the causes is the complexity of the physics-word: for example, there can be a lot of minor distortions cause by variations of pressure, temperature or traces of contamination in the raw material. When training a basic algorithm on a dataset obtained in a controlled (laboratory) environment, all those aspects are not taken into account, leading to a mismatch between prediction and empirical evidence. This issue forced to the development of advanced machine learning algorithms, called "physics-informed" which implement the partial differential equations (PDEs) governing some physical aspects of the considered process into the training algorithm, to sharply improve prediction consistency even if applied in non-lab settings. A PINN is typically composed of standard neural network layers that map input variables (for example space and time coordinates) to output variables (physical quantities like velocity, pressure, or temperature). The loss function of a PINN includes terms that ensure the satisfaction of the physical laws. It usually has two main components:

- **Data Loss**: Measures the difference between the predicted outputs and observational data.
- **Physics Loss**: Measures the difference between the predicted outputs and the solution of the PDEs. This term ensures that the network's predictions are coherent with the specified physical laws.

The training of PINNs is implemented with optimization algorithms to minimize the combined loss (data loss + physics loss). Comparing PINNs with non physic-informed machine learning model there are some advantages and some challenges to overcome, shown in the table 2.1.

Advantages	Challenges
Data efficiency: less training data required thanks to the PDEs implementation	Training can be computationally intensive ($\rightarrow$ expensive)
Better generalization of unseen scenarios	It can be hard to balance data loss and physics loss
Can handle more complex problems	Sensitive to the initial weights of the parameters

Table 2.1: Advantages and Challenges of PINNs

2.3 PINNs applications

In the process of developing PINNs able to model relatively complex physical systems it was essential to first validate and test the technology by applying PINNs to simpler and well-studied physics models. The use cases taken into consideration were:

Damped harmonic oscillator (Ben Moseley Model)

This is a popular introductory problem for PINNs because it showcases how the network is able to understand and model the loss of energy that causes the amplitude of the oscillations to decrease progressively, a setting where classical neural networks struggle. The model equations are:

- **Governing equation:**

$$m\frac{d^2u}{dx^2} + \mu\frac{du}{dx} + ku = 0 \tag{2.1}$$

  Where m is the mass of the oscillator, μ is the coefficient of friction, and k is the spring constant.

- **Initial conditions:**

$$u(0) = 1, \quad \frac{du}{dt} = 0 \tag{2.2}$$

$$\delta < \omega_0, \quad \text{with} \quad \delta = \frac{\mu}{2m}, \quad \omega_0 = \sqrt{\frac{k}{m}}$$

- **Exact solution:**

$$u(t) = e^{-\delta t}\left(2A\cos(\phi + \omega t)\right), \quad \text{with} \quad \omega = \sqrt{\omega_0^2 - \delta^2}. \tag{2.3}$$

As shown in fig 2.4, the PINN correctly learnt the behaviour of the physical model after approximately 15000 epochs.

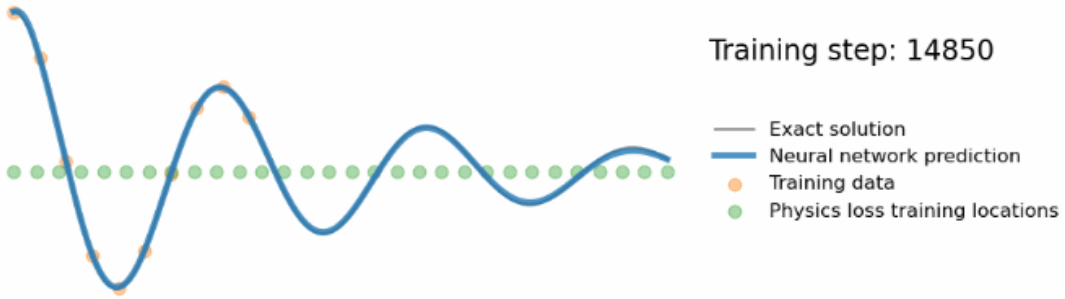

Figure 2.4: PINN accuracy compared to the exact solution.

2.3.1 1D Burgers Equation

Next, the 1D Burgers equation, fundamental PDE studied in fluid dynamics. It is widely used for tesing numerical methods due to its balance between simplicity and complexity. The governing equation has the form of:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1],\ t \in [0, 1]$$

Initial conditions and boundary condition, of Dirichlet type, follow:

$$u(-1, t) = u(1, t) = 0, \quad u(x, 0) = -\sin(\pi x).$$

The model takes as input spatial coordinates (x) and temporal coordinates (t) and gives as output the solution of the 1D Burgers Equation, i.e. the predicted

values for the velocity field u(x,t). The model output, in the form of contour plots, is then validated against the exact solutions, as shown in figure 2.5 below:

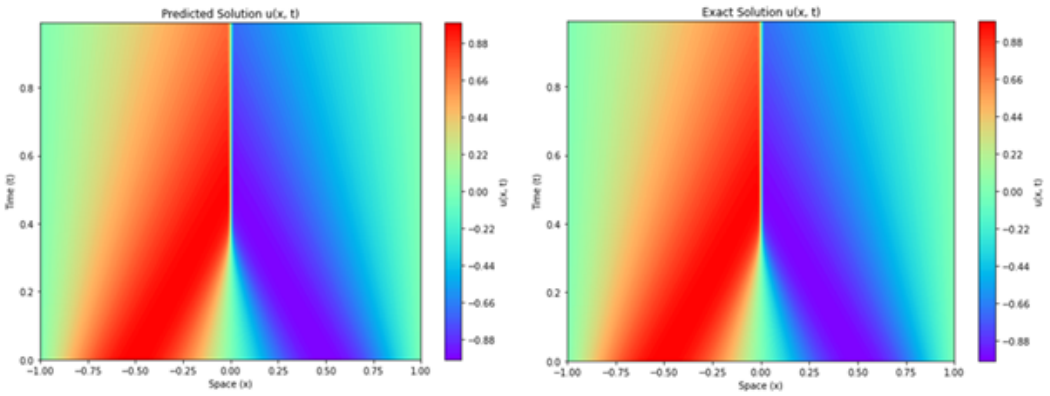

Figure 2.5: PINN accuracy compared to the exact solution.

With the following error map, a visual representation of the spatial and temporal distribution of errors between predicted and true solutions across the considered domain (in this case [-1, 1]), i.e.:

$$\text{Error}(x,t) = |u_{\text{pred}}(x,t) - u_{\text{true}}(x,t)|$$

The results is visible in the contour plot in fig 2.6 below:

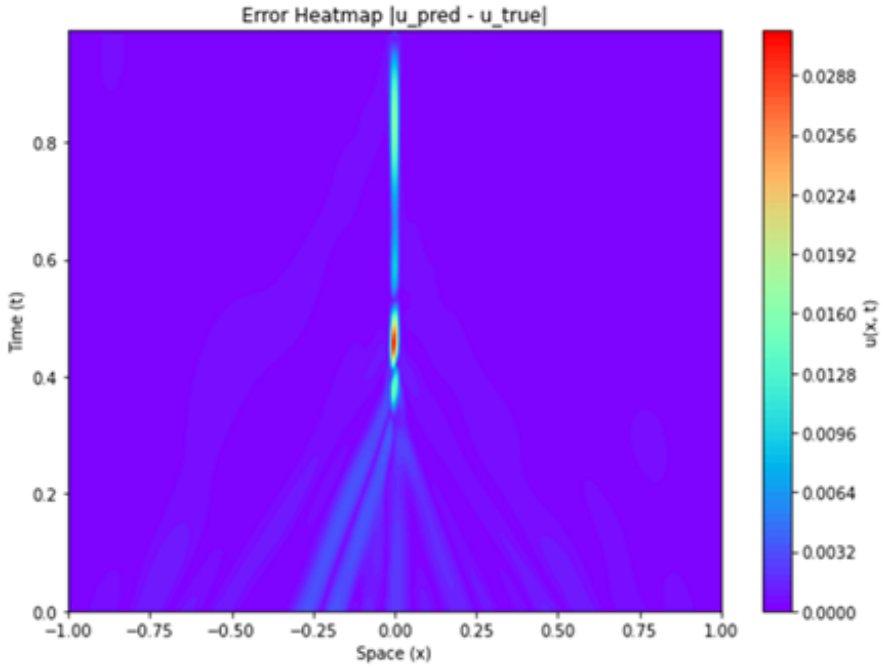

Figure 2.6: Error contourn plot for 1D Burgers Equation PINN

2.3.2 1D Heat equation

The next studied models are simpler instances of the standard 3D heat equation. The first model proposed can model the heat transfer in a 1D rod subject to:

where:

- $u(x,t)$ represents the temperature at position x and time t.
- The term $\frac{\partial u}{\partial t}$ indicates how the temperature changes over time.
- The term $\frac{\partial^2 u}{\partial x^2}$ represents the heat diffusion along the spatial dimension x.

This equation models the flow of heat in a 1D rod or bar and assumes that thermal diffusivity is normalized to 1 (i.e., $\alpha = 1$).

Initial Condition (IC)

The initial condition specifies the temperature distribution along the rod at time $t = 0$:

$$u(x,0) = \sin(\pi x)$$

This means that at $t = 0$, the temperature distribution is represented by a sine wave along the spatial domain $x \in [0, 1]$, so that all temperature values in the heat map (fig 4.2) range from 0 to 1.

Boundary Conditions (BCs)

The boundary conditions define the temperature at the ends of the rod for all times t:

$$u(0, t) = 0 \quad \text{and} \quad u(1, t) = 0$$

These conditions indicate that the ends of the rod at $x = 0$ and $x = 1$ are held at a constant temperature of 0 K. The predicted heat map in fig. 2.7 highlights how the PINN is able to capture the behaviour of the heat propagation in 1 dimension. As visible in the temperature scale, the output has been normalized in a [0,1] scale to ensure consistency during the training.

2.3.3 2D Transient Heat Equation (DeepXDE)

Moving forward the PINN model has been applied to the 2-dimensional heat equation. Compared to the previous models, coded using Pytorch library in Python to handle derivatives, this time I used a new library called DeepXDE, specifically created for PINNs and scientific machine learning, which allows for an easier implementation, particularly for boundary conditions.

Governing Equation

$$\rho c \frac{\partial T}{\partial t} - \kappa \nabla^2 T = 0 \quad \text{on } \Omega$$

where:

- ρ is the density of the material.
- c is the specific heat capacity.
- κ is the thermal conductivity.

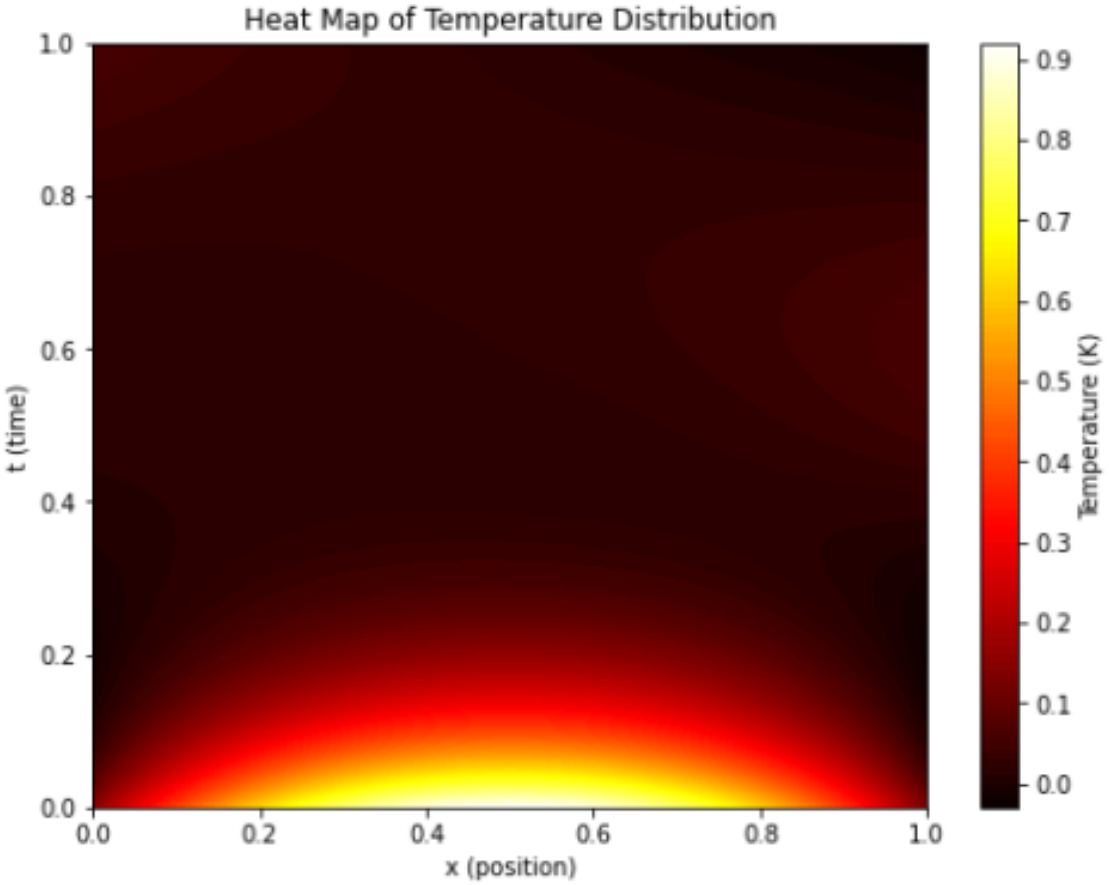

Figure 2.7: Heat map for the 1D rod

- $\nabla^2 T$ represents the Laplacian of T, it accounts for the heat diffusion across the dimensions

Initial and Boundary Conditions

- **Zero Neumann Boundary Condition** at the top and bottom edges:
  - No heat flux across these edges.

$$\frac{\partial T}{\partial n} = 0 \quad \text{on } \partial\Omega$$

- **Dirichlet Boundary Condition** at the left and right edges:

– Temperature values are fixed:

$$T(0, y, t) = 0 \quad \text{and} \quad T(1, y, t) = 30$$

– Initial temperature conditions:

$$T(x, y, 0) = \begin{cases} 0°\text{C} & \text{for } x < 0.5\,\text{m} \\ 60(x - 0.5)°\text{C} & \text{for } x \geq 0.5\,\text{m} \end{cases}$$

where $T(x, y, 0)$ indicates the starting temperature distribution in the domain:
- The temperature is 0°C for points where $x < 0.5$.
- The temperature rises until 60°C for points where $x \geq 0.5$.

Material Properties

In this model the necessary physics constant have been hard coded as follows:

- Spatial domain: $x \in [0, 1]$ and $y \in [0, 1]$
- Thermal conductivity, $\kappa = 385$ W m^{-1} C^{-1}
- Density, $\rho = 6000$ kg m^{-3}
- Specific heat capacity, $\rho c = 3.8151 \times 10^3$ N m^{-2} C^{-1}

Normalization and results

The temperature range (in this case 0-30°C) has been normalized to a 0-1 interval to improve the ability to generalize of the PINN model; after the training part it has been scaled up before plotting.

The reformulated equation is in the form:

$$\frac{\partial T}{\partial t} - \frac{\kappa}{\rho c} \nabla^2 T = 0$$

where $\frac{\kappa}{\rho c}$ represents the normalized diffusivity term.

The final output, in the form of an heat map and the loss minimization, is shown in fig 2.8 below:

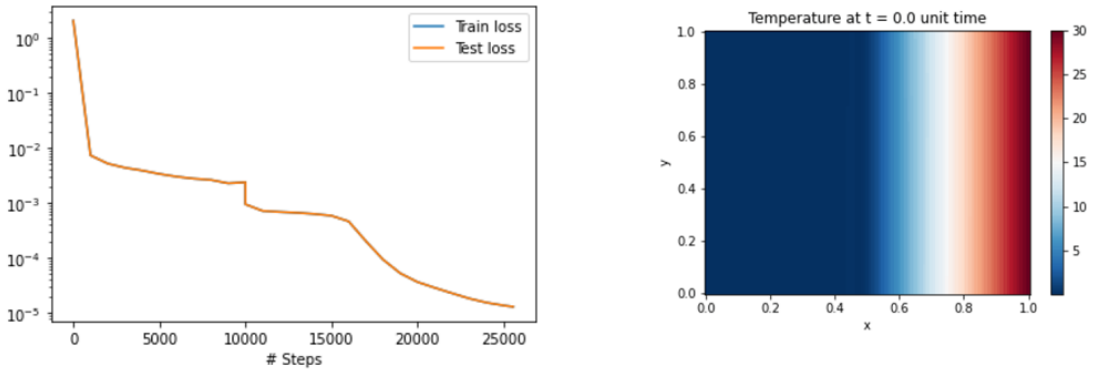

Figure 2.8: Decreasing loss value and heat map for a 2D heat equation applied on a 1x1 square

2D Transient Heat Equation (Robin BC)

The scope of the next proposed model is to test how the PINN behaves when facing a Robin boundary condition i.e. a conditions that represent convective heat transfer between the boundary and the surrounding environment, while keeping the general model the same as in previous cases.

Boundary and Initial conditions:

- **Zero Neumann BC** at left and right edges (representing insulation).
- **Robin BC** at the top and bottom edges, representing convective heat transfer with the surroundings (for example, quenching processes):

$$-\kappa \frac{\partial u}{\partial y} = h(u - T_\infty)$$

  where:

  - h is the convective heat transfer coefficient.
  - T_∞ is the ambient temperature, normalized to 0 in this model.

The output was an heat map for time segments ranging from 0-1 seconds. In fig. 4.5 it is shown the heat distribution for t=0.55 s.

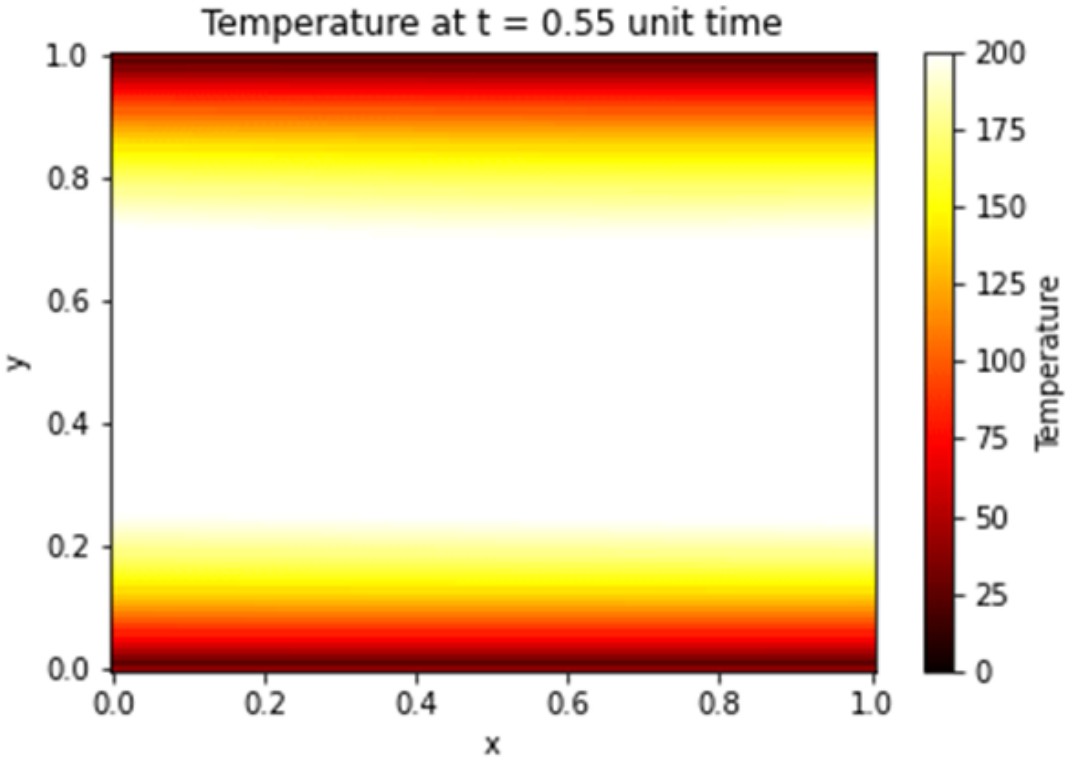

Figure 2.9: Heat map for t=0.55 seconds, with Zero Neumann BC and Robin BC.

2.4 Challenges and Limitations

Despite the promising applications for PINNs in solving forward and inverse problems described by partial differential equations (PDEs) there are some settings and limitations for which undertaking a PINN approach poses some challenges. One of the primary ones is their susceptibility to not capturing high frequency components of the solution, leading to inaccurate predictions in problems with sharp gradients or discontinuities, making PINNs less effective in modeling highly nonlinear physical systems. Another major limit is represented by computational cost and convergence challenges. In fact, training PINNs requires solving a highly nonconvex

optimization problem that can sometimes lead to slow and unstable convergence. The choice of loss function significantly affects training performance, and wrong weighting can lead to bad approximation of the underlying physics. Additionally, PINNs can struggle with complex geometries and boundary conditions, requiring an excellent understanding of PINN architecture in order to implement them and a lot of hyperparameters tuning. Also PINNs can often face difficulties in scalability and generalization. They perform well in problems with lower dimensions, while they struggle with high dimensions PDEs. All in all, harder PINNs applications require strict hyperparameter tuning, sophisticated architectures, or hybrid approaches to achieve a good accuracy and to apply this framework to complex physics environment it's mandatory to address these limitations simultaneously.

Chapter 3

Application of a PINN to model a linear elasticity problem

2D linear elasticity problems are key topics in solid mechanics, describing how materials deform under forces or displacements. It assumes a linear relationship between stresses and strains, which is valid for small deformations.

In the following problem, inspired by "Physics-informed neural networks for modeling dynamic linear elasticity" (Kag et al), a 2D cantilever beam is modeled under oscillatory displacement applied at its free end. The beam of dimensions 10x200 mm, made of a linear elastic material, is fixed at one end, with its deformation and stress components governed by linear elasticity laws. The assumed material properties are:

$$\lambda = 0.533334MPa$$

$$\rho = 0.92 * 10^{-6} kg/mm^3$$

The total simulation time is 1 second and the number of points in the FEM grid was 151201.

3.1 Governing Equations

The governing equations for linear elasticity are a system of five equations that describe the dynamic behavior of elastic materials under deformation by combining the relationships between stress, strain, and displacement fields. In particulare, three of the five equations model the stress-strain relationships (based on Hooke's law) and the remaining two enforce the conservation of linear momentum (momentum balance equations). All in all, the set of equations links the material elastic properties (λ, μ, ρ), the displacement components (u_x, u_y) and the stress components (σ_{xx}, σ_{yy}, σ_{xy}).

Stresses components

The stress components $(\sigma_{xx}, \sigma_{yy}, \sigma_{xy})$ are defined in terms of the displacement gradients (u_x, u_y) and the material properties (λ, μ), where:

- λ and μ are Lamé constants, which describe the elastic behavior of the material.
- u_x and u_y are the displacements along the x and y directions.

The equations are as follows:

$$\sigma_{xx} = (\lambda + 2\mu)\frac{\partial u_x}{\partial x} + \lambda\frac{\partial u_y}{\partial y},$$

$$\sigma_{yy} = (\lambda + 2\mu)\frac{\partial u_y}{\partial y} + \lambda\frac{\partial u_x}{\partial x},$$

$$\sigma_{xy} = \mu\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right).$$

These equations relate the stress tensor components to the strain components through the elastic properties of the material.

Momentum Balance Equations

The momentum balance equations describe the equilibrium of linear momentum in the elastic medium. They link the stress gradients to the accelerations in the x

and y directions:

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} = \rho \frac{\partial^2 u_x}{\partial t^2},$$

$$\frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{xy}}{\partial x} = \rho \frac{\partial^2 u_y}{\partial t^2}.$$

- The terms $\frac{\partial \sigma_{xx}}{\partial x}$, $\frac{\partial \sigma_{xy}}{\partial y}$, etc., represent the spatial derivatives of the stress components.
- ρ is the density of the material.
- The terms $\frac{\partial^2 u_x}{\partial t^2}$ and $\frac{\partial^2 u_y}{\partial t^2}$ represent the second time derivatives of the displacements (the acceleration components).

Boundary and Initial Conditions

Physics Informed Neural Networks can struggle if the problem it not well-posed. In this context, to improve the network accuracy, conditions on displacement and stresses components where imposed at the boundaries:

Boundary Conditions

The cantilever model is defined within a rectangular domain, where boundary conditions are imposed on all edges of the geometry to simulate physical constraints. The applied boundary conditions are classified into Dirichlet and Neumann types, each one aimed at representing a specific component of stress or displacement. Below they are listed based on the edge they are applied on:

- **Left Boundary ($x = 0$)**: At this side the cantilever is fixed and cannot mode, this constraint is modeled through a Dirichlet-type fixed boundary condition, imposing both displacement components to zero:

  $$u_x = 0, \quad u_y = 0.$$

  Furthermore, Neumann boundary condition is also imposed to the derivative of u_y respect to x representing a stress-free state for the shear stress component

which means no shear deformations on the side where the beam is fixed.

$$\frac{\partial u_y}{\partial x} = 0.$$

- **Right Boundary** ($x = L$): An oscillatory displacement is fixed for the vertical displacement u_y, modeled through a Dirichlet boundary condition as:

$$u_y = -3sin(7t),$$

Dirichlet boundary conditions are also enforced for the normal stress components:

$$\sigma_{xx} = 0, \quad \sigma_{yy} = 0.$$

- **Top and Bottom Boundaries** ($y = H \; and \; y = 0$): On the top and bottom boundaries the applied boundary conditions are of the Dirichlet-type for the stress components:

$$\sigma_{yy} = 0, \quad \sigma_{xy} = 0.$$

Initial Conditions

The initial condition enforces a state of rest for the cantilever at the beginning of the simulation, initial conditions are defined at $t = 0$ for all components of displacement and stress. These conditions are expressed as:

$$u_x = 0, \quad u_y = 0, \quad \sigma_{xx} = 0, \quad \sigma_{yy} = 0, \quad \sigma_{xy} = 0.$$

3.2 Python Implementation

The Python implementation was carried out using Deepxde, a library specifically designed for physics-informed neural networks, and Pytorch.

The first thing to do, as described by Kag et al. in the paper, is to scale the quantities in a zero to one interval. This is because the PINN is able to converge better when working using normalized data. The scaling is carried out

by dividing each quantity by their absolute maximum value which is known for physical constants and obtained through a FEM simulation for the others. Before plotting the model predictions the output are scaled back by multiplying them back for their scaling factor.

The governing equations have been implemented by computing the residuals of the momentum balance equations and the stress-strain relationships. These residuals are defined in the function called "pde" in figure 3.1, which evaluates if the network outputs satisfy the physical laws.

Then, during the training, the network minimizes the residuals of these equations, making sure that the predicted displacements and stresses are physically consistent.

```
def pde(x, f):
    E_xx = jacobian(f, x, i=0, j=1)
    E_yy = jacobian(f, x, i=1, j=1)
    E_xy = 0.5 * ((Ux_c / Y_c) * jacobian(f, x, i=0, j=1) + (Uy_c / X_c) * jacobian(f, x, i=1, j=0))
    ux_tt = hessian(f, x, i=0, j=2)
    uy_tt = hessian(f, x, i=1, j=2)

    S_xx = (Ux_c / X_c / Sxx_c) * E_xx * (2 * mu_phys + lambda_phys) + (Uy_c / Y_c / Sxx_c) * E_yy * lambda_phys
    S_yy = (Uy_c / Y_c / Syy_c) * E_yy * (2 * mu_phys + lambda_phys) + (Ux_c / X_c / Syy_c) * E_xx * lambda_phys
    S_xy = (1 / Sxy_c) * E_xy * 2 * mu_phys

    Sxx_x = jacobian(f, x, i=2, j=0)
    Syy_y = jacobian(f, x, i=3, j=1)
    Sxy_x = jacobian(f, x, i=4, j=0)
    Sxy_y = jacobian(f, x, i=4, j=1)

    momentum_x = Sxx_x + Sxy_y * (Sxy_c / Sxx_c * X_c / Y_c) - density * (X_c * Ux_c / Sxx_c / T_c / T_c) * ux_tt
    momentum_y = Sxy_x + Syy_y * (Sxy_c / Syy_c * Y_c / X_c) - density * (Y_c * Uy_c / Syy_c / T_c / T_c) * uy_tt

    sigma_xx = S_xx - f[:, 2:3]
    sigma_yy = S_yy - f[:, 3:4]
    sigma_xy = S_xy - f[:, 4:5]

    return [momentum_x, momentum_y, sigma_xx, sigma_yy, sigma_xy]
```

Figure 3.1: Code snippet of the implementation of governing equations.

The oscillation at the free side has been modeled as a Dirichlet type boundary condition by the function in fig. 3.2:

```
def oscillazione(x):
    return -3 / Uy_c * np.sin(7 * x[:, 2:3])
```

Figure 3.2: Code snippet for the oscillating side modeling.

Next, the implementation of all the boundary and initial conditions listed in the previous chapter is carried out by applying the respective constraint type and location of the respective boundary by the function in fig. 3.3:

```
ux_left_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_left, component=0)
uy_left_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_left, component=1)

duy_dx_left_bc = dde.icbc.NeumannBC(geomtime, lambda x: 0, boundary_left, component=1)

uy_right_bc = dde.icbc.DirichletBC(geomtime, oscillazione, boundary_right, component=1)

sxx_right_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_right, component=2)
syy_right_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_right, component=3)
syy_top_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_top, component=3)
syy_bottom_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_bottom, component=3)
sxy_top_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_top, component=4)
sxy_bottom_bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, boundary_bottom, component=4)

ic_ux = dde.IC(geomtime, lambda _: 0, boundary_initial, component=0)
ic_uy = dde.IC(geomtime, lambda _: 0, boundary_initial, component=1)
ic_sxx = dde.IC(geomtime, lambda _: 0, boundary_initial, component=2)
ic_syy = dde.IC(geomtime, lambda _: 0, boundary_initial, component=3)
ic_sxy = dde.IC(geomtime, lambda _: 0, boundary_initial, component=4)
```

Figure 3.3: Implementation of all the boundary and initial conditions.

To incorporate finite element method (FEM) data into the PINN, as shown in fig 3.4, `PointSetBC` is used to enforce the FEM results as boundary conditions, in fact it applies the experimental data points to μ_x, μ_y, σ_{xx}, σ_{yy}, σ_{xy} (neural network outputs) at corresponding space and time coordinates. In this case:

```
fem_ux = dde.icbc.PointSetBC(X_train, Ux_train, component=0)
fem_uy = dde.icbc.PointSetBC(X_train, Uy_train, component=1)
fem_sxy = dde.icbc.PointSetBC(X_train, Sxy_train, component=4)
fem_sxx = dde.icbc.PointSetBC(X_train, Sxx_train, component=2)
fem_syy = dde.icbc.PointSetBC(X_train, Syy_train, component=3)
```

Figure 3.4: FEM data as boundary conditions

X_{train} represents the space and time coordinates where the FEM data ($u_{x,\text{train}}$, $u_{y,\text{train}}$, $\sigma_{xx,\text{train}}$, $\sigma_{yy,\text{train}}$, $\sigma_{xy,\text{train}}$) is given as training help. By enforcing these values at the corresponding space-time location, the neural network learns to align its outputs with the FEM solutions at the specified points, incorporating additional knowledge into the training process.

Going forward with the implementation, in fig 3.5 the PINN is defined by incorporating all the boundary conditions, initial conditions, geometry and time

domain and collocation points across the boundaries and the entire domain through the `TimePDE` class. This class is used to define the dataset for the training.

```
data = dde.data.TimePDE(
    geomtime,
    pde,
    bcs,
    num_domain=domainpoints,
    num_boundary=boundarypoints,
    num_test=testpoints,
    num_initial=initialpoints
)
```

Figure 3.5: PINN data

In the code snippet, the dataset is constructed using the following components:

- **`geomtime`**: Represents the spatiotemporal domain, a 2D cantilever of 200x10mm and a time domain of 1 second.
- **`pde`**: The function representing the governing equations of the system, this defines the residuals that the network will minimize during the training.
- **`bcs`**: The full list of boundary and initial conditions, plus the constraint on the FEM data samples.
- **`num_domain`**: Specifies the number of collocation points sampled inside the domain for evaluating the PDE residuals, set to 700.
- **`num_boundary`**: Specifies the number of points specifically sampled on the boundary to enforce the boundary conditions during the training, set at 700.
- **`num_test`**: It is the number of points used to validate the accuracy of the solution, set at 200.
- **`num_initial`**: Indicates the number of points sampled at time equal to zero to enforce the initial conditions, set at 700.

The only missing steps are the definition, configuration and training of the physics informed neural network and the hyperparameter choice, shown in snippet 5.6:

```
layers = [3, [128] * 5, [128] * 5, [128] * 5, [128] * 5, 5]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.nn.PFNN(layers, activation, initializer)
resampler = dde.callbacks.PDEPointResampler(period=100)
checkpoint = dde.callbacks.ModelCheckpoint("model_iter_transient", verbose=1, save_better_only=True, period=1000)
model = dde.Model(data, net)
model.compile("adam", lr=learning_rate)
losshistory, train_state = model.train(iterations=n_epochs, callbacks=[resampler, checkpoint])
dde.saveplot(losshistory, train_state, issave=True, isplot=False)
```

Figure 3.6: Model definition, configuration and training

The network is built using the `PFNN` Deepxde class, a parallel fully-connected network which takes as input space and time coordinates and characterized by the use of independent sub-networks (in this example, 4 parallel networks, each containing 5 hidden layers of 128 neurons) to approximate each network output. The activation function is set to tanh and the weight are initialized using the "Glorot uniform" method, both well suited for problems with smooth solutions and regular domains.
The training process is done throught the Adam optimizer with learning rate set at 0.001 and trained for `n_epochs`=50000.
The `ModelCheckpoint` callback saves the model every 1000 iterations only if the model improves.
The training is improved using `PDEPointResampler` to dynamically adjust the collocation points every 100 training iterations, this method improves the performance of the model by focusing on areas where the PDE residuals are higher. Lastly, for the performance evaluation, the loss history (`losshistory`) and training state (`train_state`) are stored in a different file using `dde.saveplot`.

After the training, it is possible to obtain snapshots by using the model predict method to evaluate the trained PINN at a specific space and time coordinates, given through the 2D array `Inputs_snapshot` (fig. 3.7). The predicted values for the quantities of interest are then scaled back to original physical range before being used for plotting purposes.

```
prediction_snapshot = model.predict(Inputs_snapshot)
```

Figure 3.7: Model prediction, input is a 2D array with time and space coordinates.

Plots and Results

The implementation of the plotting section of the code is straightforward using standard Matplotlib functions and it is aimed to compare the PINN predictions with the FEM data in order to showcase results and highlight possible inaccuracies.

First, a set of time-series plots in a fixed spatial coordinates is generated by making predictions at 40 time steps, allowing us to compare the temporal evolution of fields such as U_x, U_y, and S_{xx} between the PINN and FEM models. Additionally, at a specific time slice (in this case at $t = 0.2$ s), the code plots spatial contour maps for all predicted variables (u_x, u_y, σ_{xx}, σ_{yy}, and σ_{xy}). These contour plots highligh the spatial distribution of the NN prediction quantities but also displays error contours, representing the absolute differences between the PINN and FEM solutions.

In figure 3.8 there is the PINN prediction, with a maximum value for u_x of 0.1 mm, for u_y the maximum displacement is 2.97 as the cantilever bends downwards (in fact, the contour plots shows negative results). In fig 3.10 it is shown the same plot for the FEM solutions, with identical maximum value for the u_y component and 0.9 mm for u_x.

Results interpretation

The FEM solution employed in this study has been obtained thanks to a mesh-based method in Ansys, using a mesh formed by 151201 nodes and extracting for 40 time-steps. To train the PINN, a total of 2000 FEM collocation points has been used, representing 0.033 % of the entire FEM dataset. The resulting plots are in figure 3.8, 3.9 and 3.10.

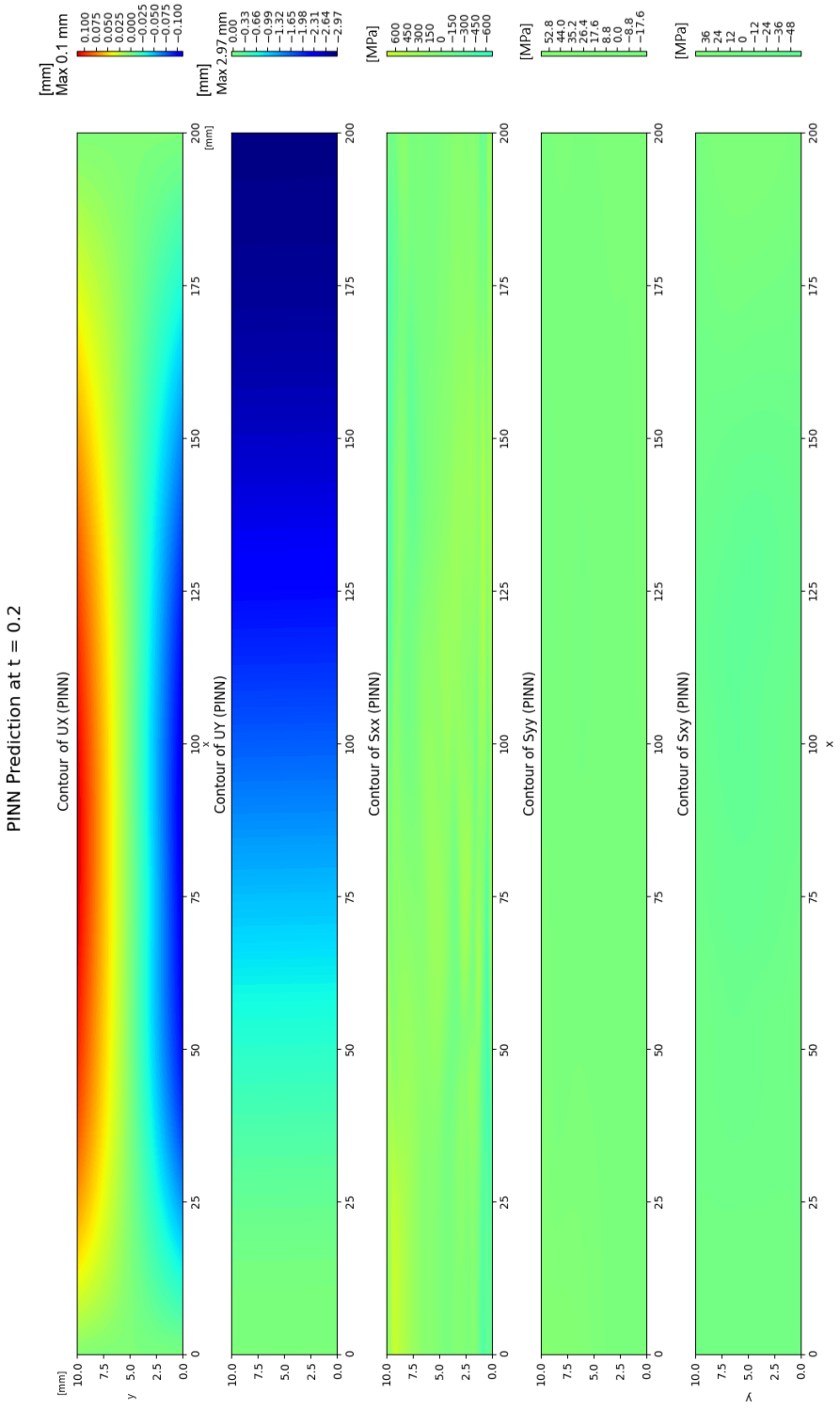

Figure 3.8: PINN prediction evaluated at 0,2 s.

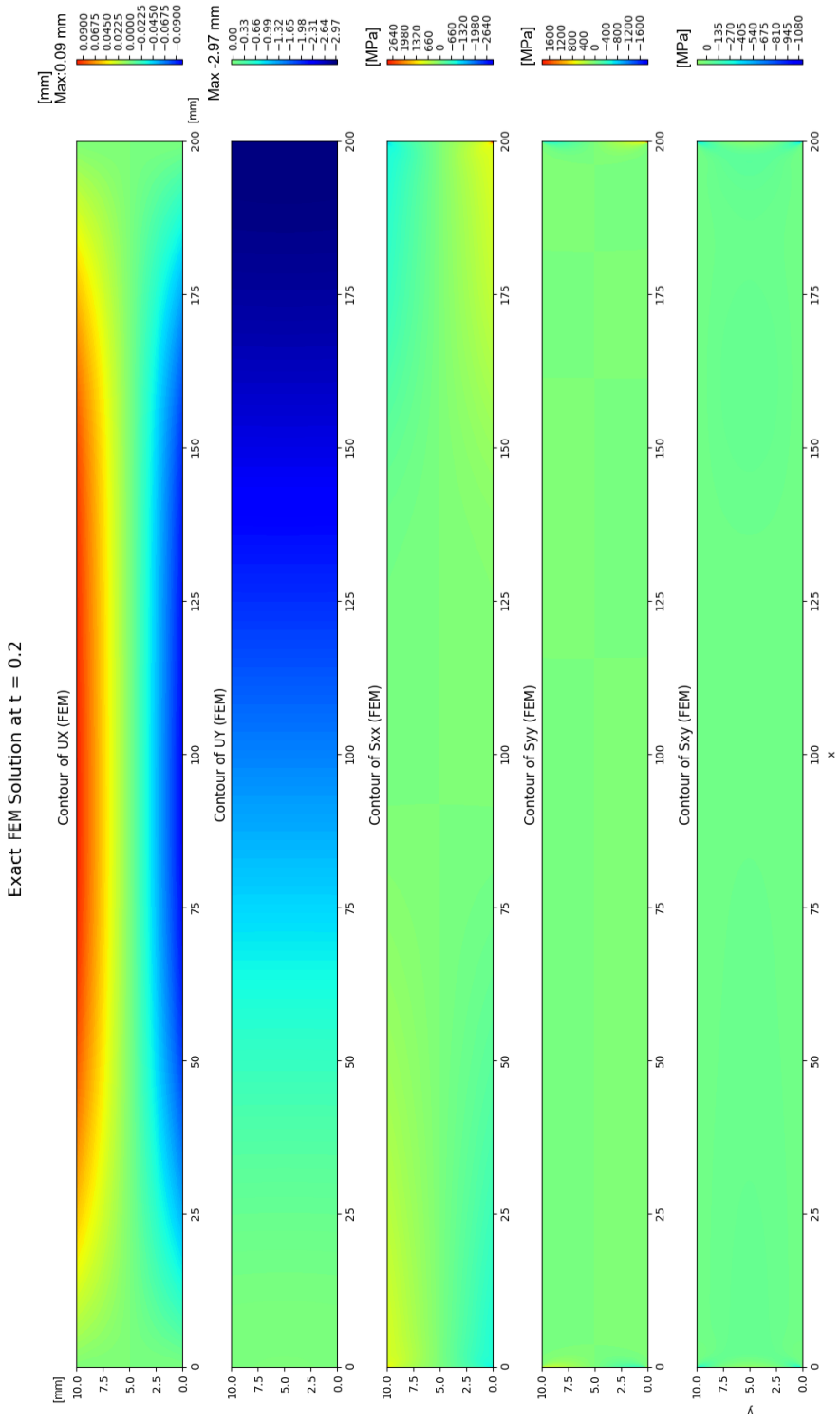

Figure 3.9: Exact FEM solution at 0.2 s.

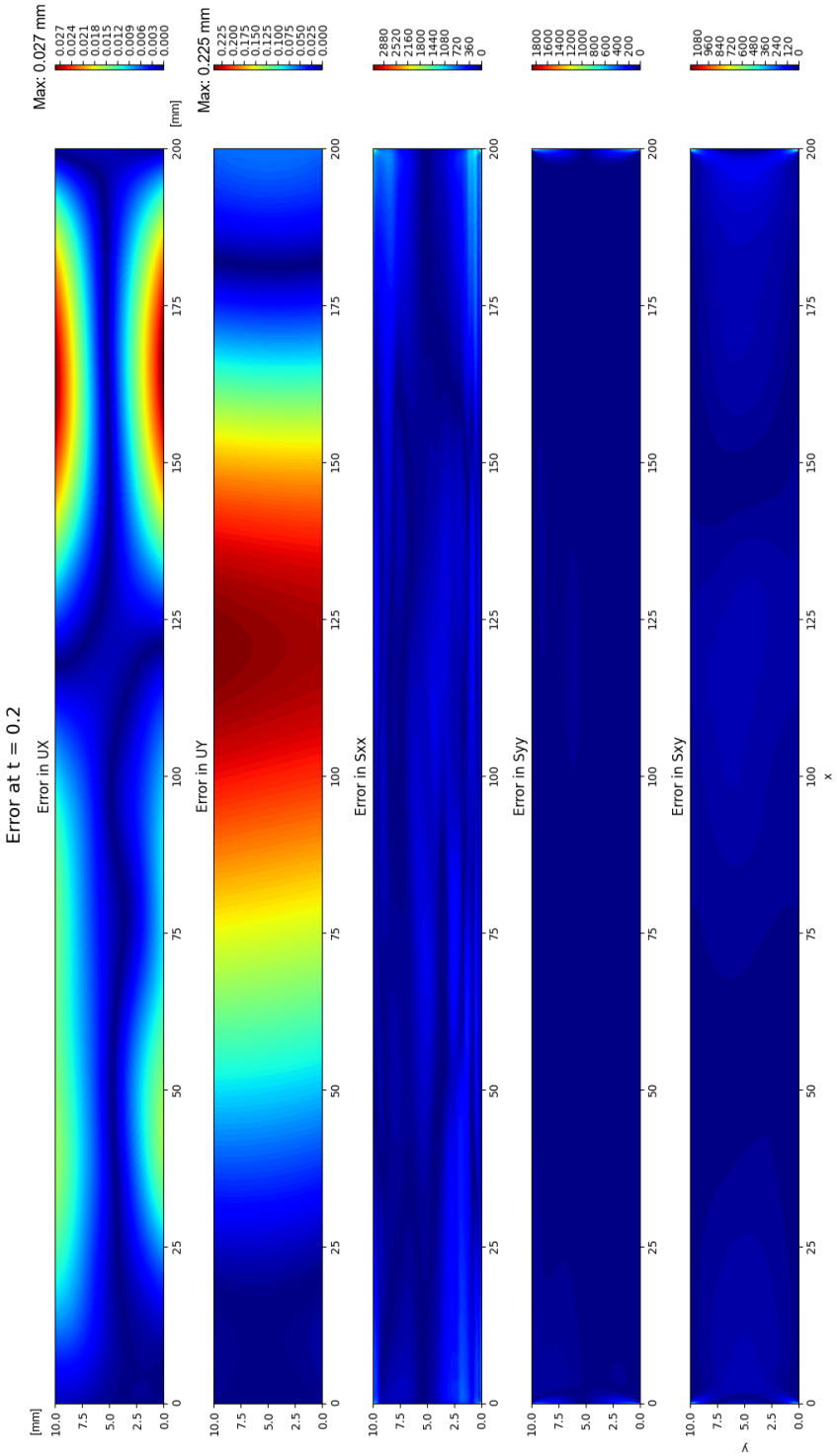

Figure 3.10: Contour graph for the PINN error evaluated against the FEM solution.

Displacement Fields (u_x, u_y)

The PINN captures the deformation pattern fairly accurately, with u_x and u_y closely resembling the FEM results. In particular, the model was able to learn the oscillation at the free side with only minor deviations in amplitude, as will be visible in the error contour plots in the next section.

Stress Fields (σ_{xx}, σ_{yy}, σ_{xy})

The behavior of the stress components is captured but comparing the FEM contour against the PINN one some discrepancies are evident. In particular, σ_{xx}, σ_{yy}, σ_{xy} all have inconsistencies in the small regions with maximum stress values at the boundaries, with regions characterized by lower stress values respect the FEM data, highlighting how PINNs can struggle in capturing localized sharp gradients. Overall, the stress fields prediction is very accurate in the rest of the domain.

Error Analysis

The error plots quantify the differences between the FEM and PINN solutions and, for each physical quantity ϕ predicted by the PINN, the error ε_ϕ has been defined as shown in formula 5.1:

$$\varepsilon_\phi = \left|\phi^{(\text{pred})} - \phi^{(\text{FEM})}\right| \times C_\phi, \tag{3.1}$$

where:

- $\phi^{(\text{pred})}$ is the value produced by the PINN, still scaled in the [0;1] interval.
- $\phi^{(\text{FEM})}$ is the (scaled) corresponding value calculated with the finite elements method.
- C_ϕ is the scaling parameter that converts the error from scaled units to the original physical units.

Looking at the contour plots, the error in u_x is nearly negligible across the most part of the domain, with slightly higher values near the fixed boundary with a

maximum value of 0.027 mm. This indicates that the PINN managed to capture the axial deformation, with minor inaccuracies in the oscillating side of the cantilever. For u_y the error is mostly in the right side of the cantilever, where the oscillatory displacement is bigger, with a maximum error of 0.225 mm, higher than the error saw for u_x, highlighting the limitation of PINNs in steep variations. This kind of error can sometimes be reduced by increasing the number of FEM data used as constraint during the training or employing different networks architectures. Analyzing the contour plots for the stress components the error in σ_{xx} is bigger near the fixed boundaries while errors in σ_{yy} and σ_{xy} are smaller and distributed more uniformly, indicating good accuracy compared with the FEM dataset. The PINN only struggled, as discussed before, with the small localized regions presenting sharp gradients near the boundaries.

Temporal comparison at fixed location

The next plotted graphs, in figure 3.11, show a transient comparison between the exact solution (FEM) and the PINN prediction evaluated a fixed spatial location inside the 2D cantilever beam, for the whole temporal domain.

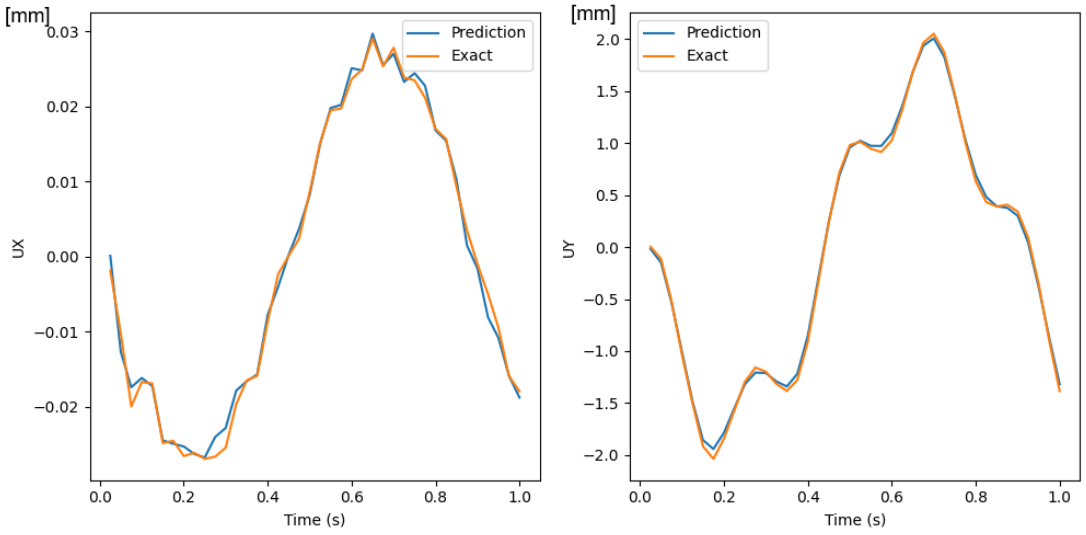

Figure 3.11: Line plots for u_x and u_y

The PINN predictions for u_x follows the FEM solution over time with only a some slight inaccuracies while for u_y the curves are nearly overlapped, with minimal distortions, confirming that the PINN learned the time dependent dynamics of the cantilever beam at the chosen spatial location.

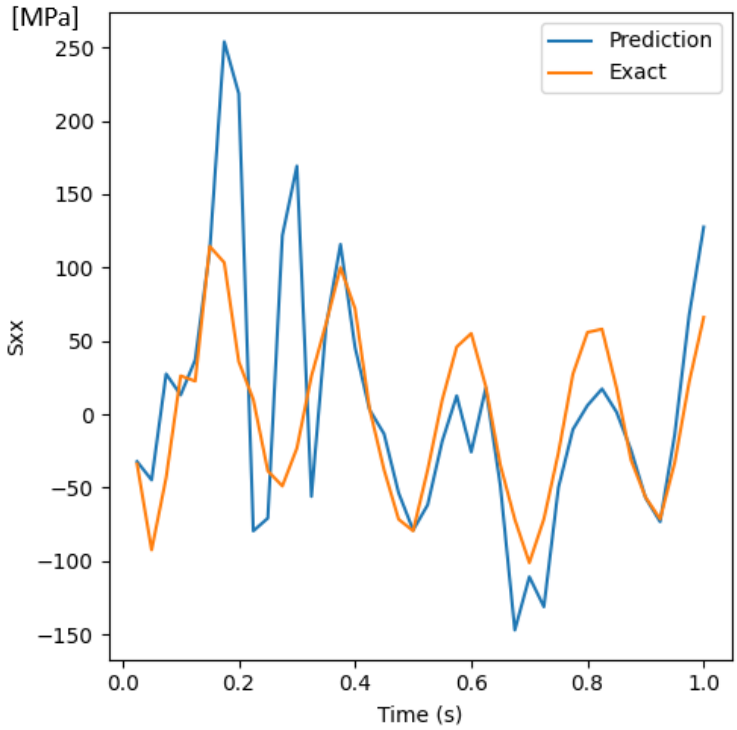

Figure 3.12: Line plot for σ_{xx}

The PINN prediction for σ_{xx}, in figure 3.12, displays lack of accuracy if compared to the FEM solution, the peaks and valleys in stress values are understimated and overstimated while the pattern is also less smooth. Despite the discrepancies, the overall trend of σ_{xx} predicted by the PINN matches the FEM solution. The frequency of the oscillations aligns fairly well with the experimental data, meaning that the PINN learned the qualitative behavior but struggled with the quantitative accuracy.

Chapter 4

Approaching multi-physics models with Resistance Spot Welding

4.1 Theoretical background

Resistance Spot Welding (RSW) is a common industrial process used for joining metal sheets, especially in industries like automotive, aerospace, and manufacturing. The process relies on the heat generated by electrical resistance at the contact point of two metal surfaces, creating a powerful bond between the materials without requiring additional filler metal.

RSW works by applying an electric current through metal electrodes, which press the metal sheets together. The electrical resistance at the contact points generates heat, melting the material and forming a small, round weld known as a nugget. Once the current stops, the molten metal solidifies, creating the joint.

The efficiency of RSW depends on several factors, such as the material properties, electrode pressure, current intensity, and welding time. Proper control of these parameters is crucial to achieving strong and consistent welds. Despite its advantages, RSW also has challenges, including nugget size variation, electrode

wear, and difficulties in welding high strength materials.

4.1.1 Governing Model

As studied by The RSW process can be accurately modeled by 3 governing equations and a set of boundary conditions (BC) and initial conditions (IC):

Fourier Equation (Before Fusion, Solid Phase)

Fourier's law describes the heat conduction from the weld zone into surrounding materials, ensuring the heat spreads out from the point of highest temperature (weld zone).

$$\frac{q_s'''}{k_s} + \nabla^2 T_s = \frac{1}{\alpha_s}\frac{\partial T_s}{\partial t} \tag{4.1}$$

where

- q_s''' is the volumetric heat generation rate in the solid phase,
- k_s is the thermal conductivity in the solid phase,
- T_s is the temperature in the solid phase,
- $\alpha_s = \frac{k_s}{\rho_s c_{ps}}$ is the thermal diffusivity in the solid phase,
- ρ_s is the density in the solid phase,
- c_{p_s} is the specific heat capacity in the solid phase.

Maxwell-Cattaneo Equation (After Fusion, Liquid Phase) The Maxwell-Cattaneo equation improves upon Fourier's law by considering the time lag in the propagation of heat. In thermal processes where temperature changes fast, like RSW, this model is more accurate.

$$\frac{q_l'''}{k_l} + \nabla^2 T_l = \frac{1}{\alpha_l}\frac{\partial T_l}{\partial t} + \frac{1}{c_l^2}\frac{\partial^2 T_l}{\partial t^2} \tag{4.2}$$

where

- q_l''', k_l, and T_l are as defined above for the liquid phase,
- c_l is the speed of propagation of thermal waves in the liquid phase.

Stefan Condition (During Fusion) The Stefan condition governs the phase change between solid and liquid during the nugget formation. The equation relates the rate of heat flow through the solid and liquid regions to the speed of the solid-liquid interface during the welding process:

$$k_s \frac{\partial T_s}{\partial z} - k_l \frac{\partial T_l}{\partial z} = \rho L \frac{d\xi(t)}{dt} = pLv(t) \tag{4.3}$$

where

- k_s and k_l are the thermal conductivities in the solid and liquid phases, respectively,
- T_s and T_l are the temperatures in the solid and liquid regions, respectively,
- ρ is the density,
- L is the latent heat of fusion,
- $\xi(t)$ is the position of the solid-liquid interface as a function of time,
- $v(t)$ is the velocity of the interface.

Boundary Conditions

The boundary conditions define how the temperature behaves at the boundaries of the welding region, including the electrode surfaces and edges of the metal sheets.

Dirichlet Boundary Condition (First Kind)

This boundary condition specifies a fixed temperature at the boundary. For example, the temperature of the electrodes can be treated as constant.

$$T(x, y, z_0, t) = T_{\text{electrode}} \tag{4.4}$$

where

- $T_{\text{electrode}}$ is the fixed temperature at the boundary (e.g., the electrode surface).

Neumann Boundary Condition (Second Kind)

This boundary condition specifies the rate at which heat is leaving or entering the system at the boundary. For example, at the metal-electrode interface, the heat generated in the welding zone is partially removed by the electrodes.

$$-k_s \frac{\partial T}{\partial z} = q''_w = F(r,t) \tag{4.5}$$

where

- q''_w is the heat flux at the boundary,
- $F(r,t)$ represents the variation of heat flux with position r and time t.

Convective Boundary Condition (Third Kind)

This condition describes convective heat transfer from the surface of the electrodes to the cooling water.

$$-k_s \frac{\partial T}{\partial z} = h(T - T_{\text{cool}}) \tag{4.6}$$

where

- h is the convective heat transfer coefficient,
- T_{cool} is the temperature of the cooling medium (e.g., water).

Fourth Kind (Conjugate Boundary Condition)

$$q'''_s + \nabla^2 T = \frac{1}{\alpha_s}\frac{\partial T}{\partial t} + \frac{1}{c_s^2}\frac{\partial^2 T}{\partial t^2} \tag{4.7}$$

where this condition accounts for both materials' heat transfer properties at the interface (e.g., metal sheet and electrode).

Initial Conditions

Initial Condition for Fourier Equation (Before Fusion, Solid Phase)

$$T(x, y, z, 0) = T_0 \tag{4.8}$$

where

- T_0 is the initial temperature distribution, often set to a constant or a known function.

Initial Condition for Maxwell-Cattaneo Equation (After Fusion, Liquid Phase)

$$T(x, y, z, 0) = T_0 \tag{4.9}$$

where

- T_0 is the initial temperature distribution, similar to the Fourier equation but applied after fusion.

4.1.2 Challenges on RSW modelilng with a traditional PINN

The analytical model proposed by Sun et el. in "Mathematical Modeling of Resistance Spot Welding" (2019) poses significant challenges in a PINN implementation, in particular:

- Stefan conditions add an important amount of complexity due to the moving boundary problem at the solid-liquid interface, with sharp gradients at the interface which PINNs struggle to learn.

- Parameters like thermal conductivity, density, specific heat, and latent heat change with temperature and phase (solid-liquid), requiring non-linear and temperature-dependent formulations.

- Realistic boundary conditions, such as heat flux on electrodes or convection with cooling water, are hard to measure and model accurately.

- The assumptions for initial conditions (in this case, the uniform temperature) oversimplify real-world scenarios where pre existing thermal gradients exist, leading to inaccurate predictions when compared to actual experimental data.

- Solving coupled PDEs with boundary and phase transition conditions in PINNs often leads to instabilities, especially in high resolution simulations required for complex physics scenarios like RSW, leading to results that do not make physical sense.

- Discrepancies between model predictions and measurements can arise from phenomena not accounted for, such as material impurities and the environmental atmosphere.

Moreover, RSW involves not just heat transfer but also electrical currents, mechanical stresses, and phase transformations and fully integrating these interactions into a single model is very complex and resource intensive.

Addressing these challenges requires advanced architecture, high-quality experimental data, and potentially hybrid models that allow to combine physics based and data driven approaches.

To overcome the challenges in the aim of this thesis work, we developed a parametric model to describe one of the main variables of interest, i.e. the nugget growth.

4.2 PINN for Modeling 2D Heat Diffusion with a Gaussian Distributed Heat Source

Due to the challenges discussed in the previous section, implementing a Physics-Informed Neural Network for the resistance spot welding process was not feasible. The challenges associated with accurately modeling the multi-physics interactions, including material phase changes, interactions at the contact points between metal plates and electrodes, and dynamic boundary conditions, made the direct application of PINNs highly complex.

As an alternative, we developed a PINN model to simulate heat diffusion in a metal plate, in two dimensions, where the heat source is modeled as a Gaussian distribution centered in a rectangular domain. This approach is aimed at approximating the Joule heating effect, key in the RSW process, focusing specifically on the thermal diffusion aspects. While this simplification does not capture the full complexity of the RSW process, it allows us to analyze the ability of PINNs to model heat transfer scenarios under conditions compatible with resistance spot welding.

The physical model is similar to the standard 2D heat equation discussed in chapter 2 but implements some changes in the boundary conditions and for the presence of an heat source.

4.2.1 Physical Model

The governing equation for the transient heat conduction with an internal heat source is given by

$$\rho c \frac{\partial T}{\partial t} - \nabla \cdot (\kappa \nabla T) = Q(x, y, T) \quad \text{in } \Omega.$$

Where:

- Ω: The spatial domain of the problem.
- ρ: Density of the material.

- c: Specific heat capacity.
- κ: Thermal conductivity.
- $T(x, y, t)$: Temperature at spatial coordinates (x, y) and time coordinate t.
- $Q(x, y, T)$: It is the internal heat generation term, defined, for a gaussian heat distribution, by

$$Q(x, y, T) = I^2 R_0 \Big[1 + \alpha \left(T - T_0\right)\Big] \exp\left(-\frac{(x - x_0)^2}{2\sigma_x^2} - \frac{(y - y_0)^2}{2\sigma_y^2}\right),$$

  - I: Applied current.
  - R_0: Resistance value at initial temperature.
  - α: Temperature coefficient of resistance.
  - T_0: Initial temperature.
  - (x_0, y_0): Center coordinates of the Gaussian heat source.
  - σ_x, σ_y: Standard deviations in the x and y directions, used to model the spread components of the Gaussian distribution, both set to 0.1 to have a narrower spread.

Initial conditions The initial condition is simply a constant temperature across the whole domain at the start of the simulation, in order to represent a room temperature metal plate:

$$T(x, y,0) = 20, \quad \forall\, (x, y) \in \Omega.$$

Where:

- Ω: The entire spatial domain of the problem, in this case it is represented by a rectangular metal plate.
- $T(x, y, t)$: Temperature at the spatial point with coordinates (x, y) at time t.
- 20: The initial temperature.

Boundary conditions

The chosen BCs are Dirichlet type boundary conditions on all four edges of the domain to model a large welded plate. These conditions ensure that the temperature is maintained at room temperature, representing the idea that the metal plate at the edges is able to dissipate heat efficiently.

$$T(x, y, t) = 20 \quad \text{for all } (x, y) \in \partial\Omega \text{ and } t \geq 0.$$

Where:

- $T(x, y, t)$: Temperature at a spatial point (x, y) at time t.
- 20: The value at which the temperature is fixed at the boundaries.
- $\partial\Omega$: The boundary of the spatial domain Ω.
- $t \geq 0$: The time interval during which this condition is applied, in this case across the whole simulation.

4.2.2 DeepXDE implementation

In this section will be provided the relevant code snippets and the results discussion.

```
def pde(X, T):
    I = 1800
    R0 = 1
    alpha = 0.004
    T0 = 20/max_T
    x0 = 0.5
    y0 = 0.25
    sx = 0.1
    sy = 0.1

    R_t = R0 * (1 + alpha * max_T*(T - T0))
    Q = I**2 * R_t
    heat_source = Q * exp(-(((X[:, 0] - x0) ** 2) / (2 * sx ** 2) + ((X[:, 1] - y0) ** 2) / (2 * sy ** 2)))
    dT_xx = dde.grad.hessian(T, X,i=0, j=0)
    dT_yy = dde.grad.hessian(T, X, i=1, j=1)
    dT_t = dde.grad.jacobian(T, X, i=0, j=2)

    return (rhoc * dT_t)/(Qref*time_max)*max_T - (kap * (dT_xx + dT_yy))/Qref*max_T - heat_source/Qref
```

Figure 4.1: Governing equations of the model.

The governing equation for the heat diffusion problem is the 2D heat equation (fig 4.1) which also includes a heat source modeled as a Gaussian function. The

resistance (R_t) varies with temperature, affecting the Joule heating term:

$$R_t = R_0(1 + \alpha \cdot (T - T_0)) \tag{4.10}$$

where R_0 is the initial resistance and α is the temperature coefficient. The heat source Q is computed with the formula:

$$Q = I^2 R_t \tag{4.11}$$

distributed (in the space domain) using a Gaussian function centered at (x_0, y_0):

$$\text{Heat source} = Q \cdot \exp\left(-\frac{(x - x_0)^2}{2s_x^2} - \frac{(y - y_0)^2}{2s_y^2}\right) \tag{4.12}$$

As discussed before, the initial temperature distribution is modeled, in fig. 4.2 as a Gaussian distribution centered at $(0.5, 0.25)$ with a standard deviation $\sigma = 0.05$:

$$T(x, y, t = 0) = T_{\text{center}} \exp\left(-\frac{(x - 0.5)^2 + (y - 0.25)^2}{2\sigma^2}\right) \tag{4.13}$$

This implementation's scope is to ensure that the initial temperature distribution decays moving away from the center.

```
def init_func(X):

    x, y = X[:, 0:1], X[:, 1:2]
    sigma = 0.05
    return t_center * np.exp(-((x - 0.5)**2 + (y - 0.25)**2)/(2*sigma*sigma))
```

Figure 4.2: Gaussian distribution for heat profile.

The spatial domain is defined as a rectangular region, imitating a metal plate, with a time dependent boundary condition. In all 4 sides of the spatial domain, the boundary conditions are of the Dirichlet type, implemented as shown in fig. 4.3.

$$T(x = 0, y, t) = t_1, \tag{4.14}$$

$$T(x = 1, y, t) = t_2, \tag{4.15}$$

$$T(x, y = 0, t) = t_3, \tag{4.16}$$

$$T(x, y = 0.5, t) = t_4. \tag{4.17}$$

The initial condition is set as:

$$T(x, y, t = 0) = \frac{20}{\max_T} \tag{4.18}$$

Where (max_T) is the maximum temperature reached in the numerical simulation.

```
geom = dde.geometry.Rectangle([0.0, 0.0], [1.0, 0.5])
timedomain = dde.geometry.TimeDomain(0.0, end_time)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

bc_l   = dde.DirichletBC(geomtime, lambda X: t1, l_boundary)
bc_r   = dde.DirichletBC(geomtime, lambda X: t2, r_boundary)
bc_up  = dde.DirichletBC(geomtime, lambda X: t3, up_boundary)
bc_low = dde.DirichletBC(geomtime, lambda X: t4, down_boundary)

ic     = dde.IC(geomtime, lambda X: 20.0/max_T, boundary_initial)
```

Figure 4.3

The PINN model is trained using a dataset obtained from numerical simulations. The dataset consists of spatial and temporal coordinates (t, x, y), paired with the corresponding temperature values. For the training, different sets of points are sampled:

- Domain Points: 1000 points sampled in the computational domain.
- Boundary Points: 2000 points sampled on the domain boundary.
- Initial Condition Points: 700 points at the initial time step.
- Test Points: 700 points used for evaluation.

```
data = dde.data.TimePDE(
    geomtime,
    pde,
    [observe_T, bc_l, bc_r, bc_up, bc_low, ic],
    num_domain=1000,
    num_boundary= 2000,
    num_initial=700,
    num_test = 700,
```

Figure 4.4: Enter Caption

The neural network used for the PINN model, shown in fig. 4.5, is a feedforward neural network with the following architecture:

- Input layer: 3 neurons (x, y, t)
- Hidden layers: 5 layers with 64 neurons each, using a `tanh` activation function
- Output layer: 1 neuron representing temperature T

The loss function is weighted across different constraints, in particular the PDE loss weight is set to 1 while all the others are set to 10, to improve the ability of the model to satisfy BCs and ICs.

Training is performed using Adam with a learning rate of 1×10^{-3}. The training is performed for 30000 epochs, with checkpointing and resampling of PDE points every 100 iterations.

```
layer_size = [3] + [64]*5 + [1]
activation_func = "tanh"
initializer = "Glorot uniform"
net = dde.maps.FNN(layer_size, activation_func, initializer)
model = dde.Model(data, net)
lr = 1e-3
loss_weights = [1, 10, 10, 10, 10, 10,10]
epochs = 30000
optimizer = "adam"
model.compile(optimizer, lr=lr,loss_weights=loss_weights)
checker = dde.callbacks.ModelCheckpoint("model/model1.ckpt",verbose=1, save_better_only=True, period=1000)
resampler = dde.callbacks.PDEPointResampler(period=100)

losshistory, train_state = model.train(epochs=epochs, callbacks=[resampler,checker])
```

Figure 4.5: Model definition and configuration

4.2.3 Results

Figure 4.6 presents the FEM solution at $t = 0$ s and $t = 100$ s. The temperature field is set at 20°C at the initial time step and it is heated over time following a gaussian distribution.

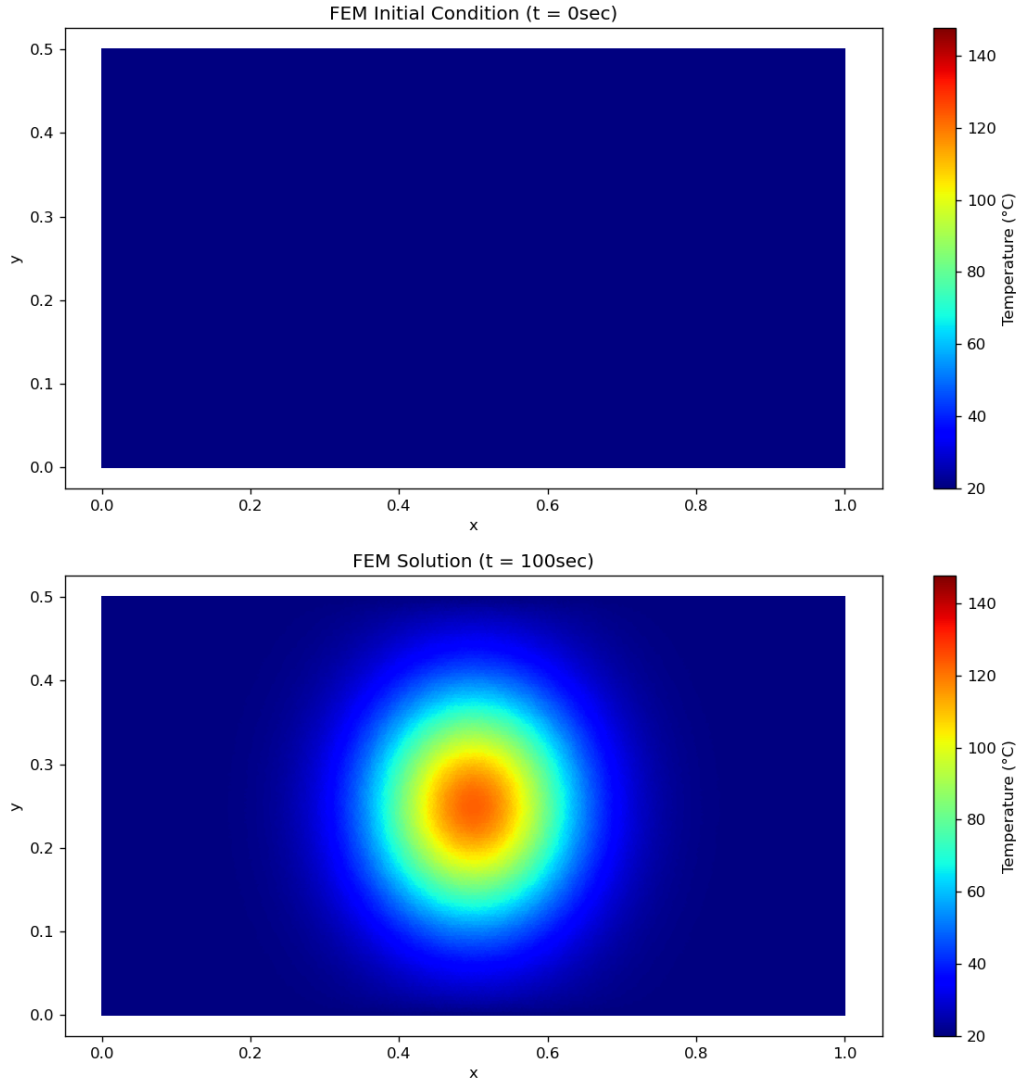

Figure 4.6: Heat map based on numerical simulation data.

Figure 4.7 illustrates the predicted temperature field at $t = 0$ s and $t = 100$ s. At first, the temperature is uniform at 20°C, consistent with the simulation data.

Over time, heat spreads from the center, forming a Gaussian distribution. The PINN effectively captures the expected diffusion pattern.

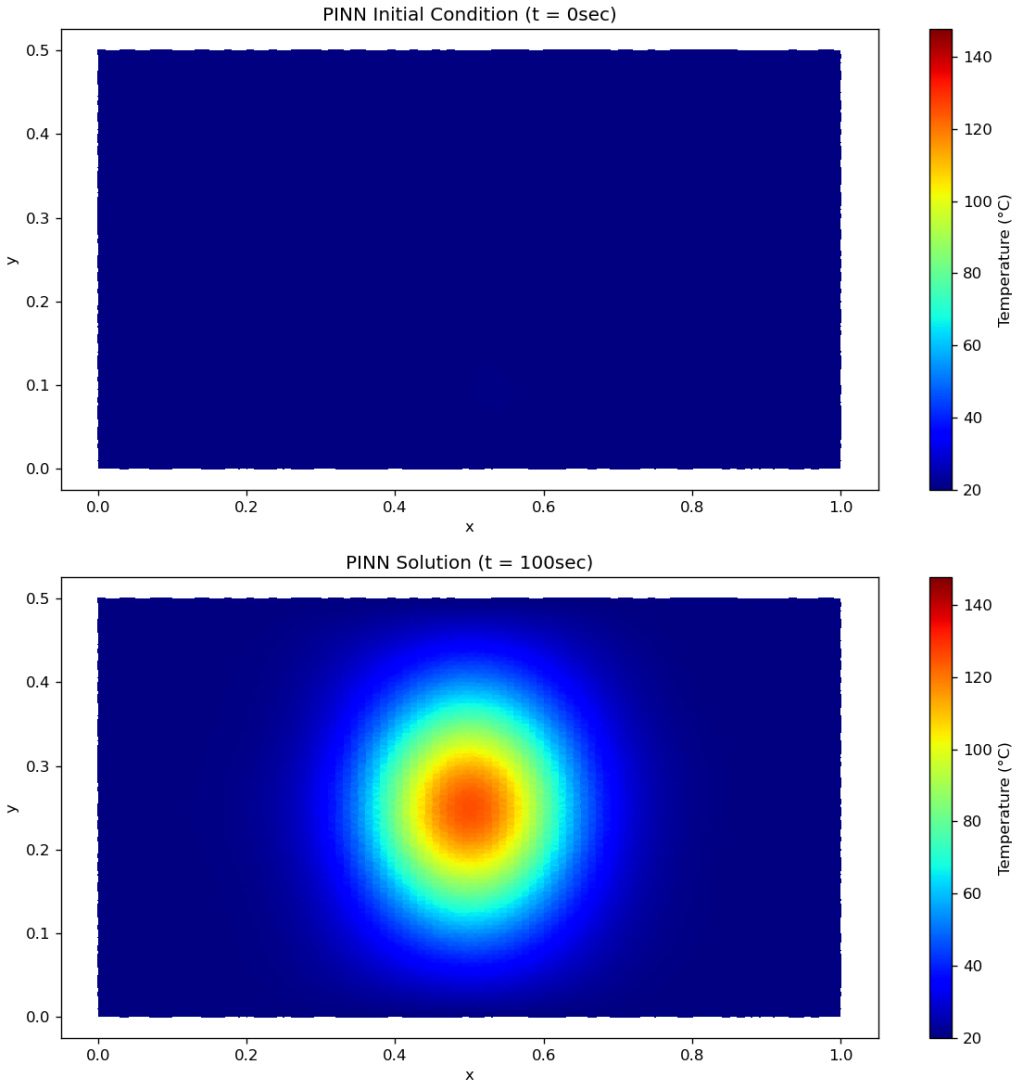

Figure 4.7: PINN prediction.

The error maps in figure 4.8 shows the difference between the PINN predicted and reference temperature distributions for the 2D scenario. At $t = 0$ sec, the error is minimal across the domain, indicating a good initial approximation. At time step $t = 100$ sec, the error remains low, within a few degrees Celsius but is more

evident near the heat source. The pattern indicates a deviation which is likely due to the PINN difficulty in capturing steep temperature gradients. Despite this, the overall accuracy is good.

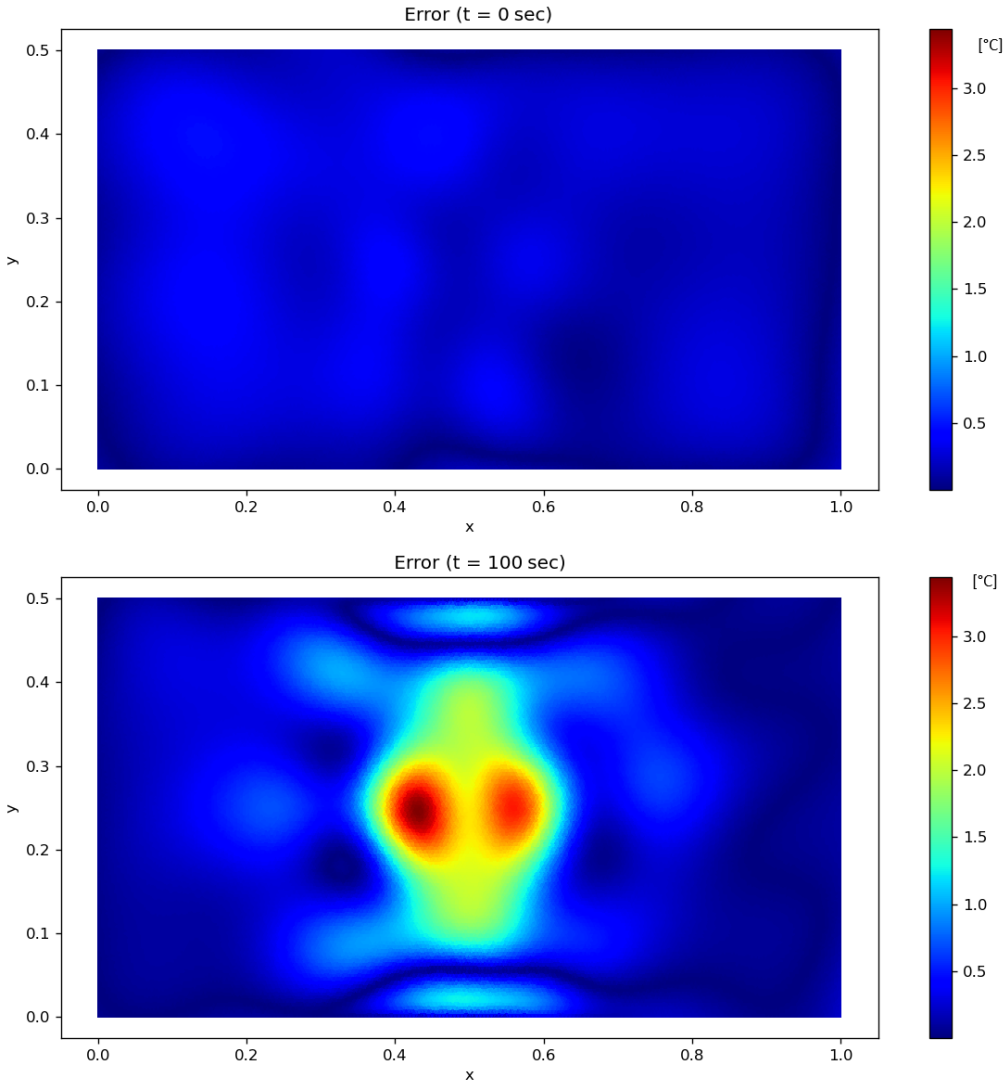

Figure 4.8: Error map for the PINN solution compared against FEM results.

Figure 4.9 shows the temperature evolution at the central point with coordinates $(x = 0.5, y = 0.25)$ over time. The PINN solution follows the validation dataset, confirming the accuracy of the model.

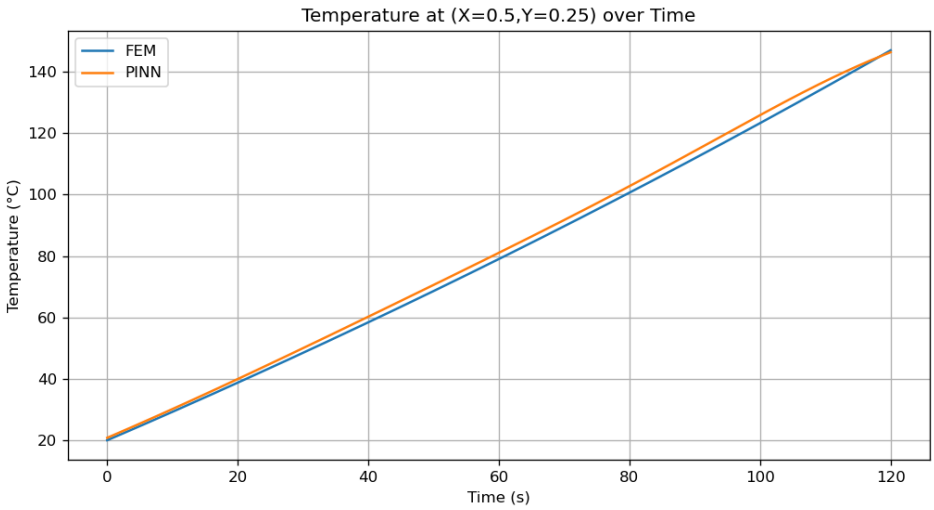

Figure 4.9: Line plot at spatial coordinates (0.5;0.25).

All in all, the PINN model successfully approximates the heat diffusion process, showing good agreement with the simulated dataset reference. This demonstrates the capability of PINNs to model transient heat diffusion problems.

4.3 Parametrization with informed ML to estimate nugget growth

In the final section of our work, we developed an ordinary differential equation (ODE) to model the nugget growth during resistance spot welding (RSW) that depends on three key parameters.

The analytical solution of this ODE provides an explicit formula for the nugget diameter, which is our quantity of interest. In order to estimate the three parameters, we adopt a hybrid Physics-Informed Machine Learning (PIML) approach. Specifically, we use a neural network to estimate the values of these parameters based on experimental data collected in the labs of Politecnico di Torino.

Once the neural network has provided its estimates, we substitute them into the analytical solution of the ODE to compute the final nugget diameter, which is our primary quantity of interest.

This method combines an analytical physics-based model with a data-driven neural network for parameter estimation, thus, by integrating previous knowledge in the form of an ODE, it falls under the umbrella of PIML according to "Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems" by Rueden et el. (2023) . However, it differs from typical PINNs implemented in chapters 3 and 4, since the neural network is not used to approximate the solution directly, but rather to parameterize an existing analytical solution.

4.3.1 ODE Model

The nugget diameter growth in resistance spot welding can be modeled by the following ordinary differential equation:

$$\frac{dD}{dt} = k_1 \frac{I^2}{P^\alpha} - k_2 D \tag{4.19}$$

where:

- D is the weld nugget diameter at time t.
- I is the welding current (in amperes).
- P is the applied pressure (force per unit area).
- k_1 and k_2 are material- and geometry-dependent constants.
- α is an empirical factor that describes how pressure affects resistance and heat generation.

More in general, the terms represent:

- **First Term** ($k_1 \frac{I^2}{P^\alpha}$):
  - Represents heat generation due to Joule heating (I^2R).
  - Pressure (P) reduces contact resistance, hence the inverse relationship.
- **Second Term** ($-k_2 D$):

– Accounts for heat dissipation and the slowing down of nugget growth as it gets larger.

The general solution to this first-order linear differential equation, implemented in the code, is:

$$D(t) = \frac{k_1 I^2}{k_2 P^{\alpha}} \left(1 - e^{-k_2 t}\right) \tag{4.20}$$

where:

- The first term $\frac{k_1 I^2}{k_2 P^{\alpha}}$ represents the steady-state nugget size.
- The term $e^{-k_2 t}$ models the time dependent behavior.

Implementation

The model is built using Keras, where:

- k_1, k_2, and α are the trainable parameters, initialized to 1.
- The forward pass computes the equation using Tensorflow operations.
- Input features are welding current (I), pressure (P), and duration (t), while the output is the predicted nugget diameter (D).

The model is trained on the dataset using Mean Squared Error (MSE) as the loss function and the Adam optimizer with a learning rate of 0.001 and the training process runs for 5000 epochs.

Once trained, the model extracts the learned values of k_1, k_2, and α, it can then be used to predict nugget diameters for new welding conditions.

After training, the parameters values learned by the model are:

$$k_1 = 0.4108, \quad k_2 = 1.4376, \quad \alpha = 0.6385$$

Overall, the model predictions for the nugget diameter values accurately capture the general trend of the experimental data, following the expected expected physical behavior but it is not highly sensitive to small variations in the input parameters.

Chapter 5

Conclusions and future work

As shown in this work, physics informed neural networks offer a promising alternative to traditional numerical methods in industrial applications, in particular for modeling physical processes that are complex. Their main advantages include:

5.1 Benefits and Challenges of PINNs approaches in industrial context

- **Data efficiency:** As discussed, PINNs leverage known physics equations and this reduces the dependence on large datasets, which is beneficial when experimental data is limited or very hard to obtain.
- **Good framework for both forward and inverse problems:** In this work PINNs implementation involved forward problems but they are also effective to discover PDEs and learn the unknown parameters, making them valuable for inverse problems.

Anyway several challenges can limit the PINN utilization in industrial scenarios:

- **Computational cost:** Training PINNs for very complex problems can require significant computational power and is often slower than FEM for standard problems.

- **Optimization complexity:** PINNs training effectiveness and optimization are affected by the network architecture, hyperparameters and loss weights, making them less practical to develop compared to the setup of a standard FEM software.
- **Accuracy and reliability:** As shown in the previous sections, PINNs can approximate solutions fairly well but it's very challenging to ensure the same accuracy of classical numerical methods.

All in all, in industrial scenarios, PINNs are better suited for problems where conventional numerical solver can struggle, such as inverse problems, or cases where mesh based approaches are too expensive by a computational point of view. Also, being PINNs an emerging and immature technology, in the current times they cannot replace FEM in most engineering applications but can provide a complementary tool if used strategically.

5.2 When to Use PINNs vs. FEM: Considerations and Analytical Solutions

In order to choose between PINNs and FEM simulations it is essential to evaluate the problem characteristics and any available dataset. FEM simulations are generally preferred when it's possibile to have a well structured mesh, providing very accurate numerical solutions. However, PINNs offer advantages if dealing with more complex geometries, sparse data or problems where standard meshing methods are computationally very expensive. Moreover, if an analytical solution exists, it is always the preferred choice due to its exactness and computational efficiency.

Bibliography

[1] Jason Brownlee. *What is the Difference Between Test and Validation Datasets?* Machine Learning Mastery blog. Accessed: 2017-10-12. 2017. URL: `https://machinelearningmastery.com/difference-test-validation-datasets/` (cit. on p. 4).

[2] Sun et al. *Mathematical Modeling of Resistance Spot Welding*. Research article. 2019.

[3] Xle et al. *3D Temperature Field Prediction in Direct Energy Deposition of Metals Using Physics-Informed Neural Network*. Conference paper. 2021.

[4] Mazuga et al. *Physics Informed Neural Network Code for 2D Transient Problems (PINN-2DT) Compatible with Google Colab*. Preprint. 2023.

[5] Sina Amini Niaki et al. *Physics-Informed Neural Network for Modelling the Thermochemical Curing Process of Composite-Tool Systems During Manufacture*. Technical report. 2021.

[6] Varghese et al. *Physics-Informed Neural Network for Concrete Manufacturing Process Optimization*. Research study. 2024.

[7] Zhung et al. *Machine Learning for Metal Additive Manufacturing: Predicting Temperature and Melt Pool Fluid Dynamics Using Physics-Informed Neural Networks*. Conference proceedings. 2021.

[8] Nazari et al. *Physics-Informed Neural Networks for Modeling Water Flows in a River Channel*. Environmental modeling study. 2024.

[9] Cai et al. *Physics-Informed Neural Networks for Heat Transfer Problems*. Engineering research paper. 2021.

[10] W. A. Strauss. *Partial Differential Equations*. Second. 2008.

[11] Moseley. *So What Is a Physics-Informed Neural Network?* Blog post. 2024. URL: https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/.

[12] Joris C. *Physics-Informed Machine Learning (PIML)*. Medium article. Accessed: Feb 2024. 2024. URL: https://medium.com/@joris.c/physics-informed-machine-learning-piml-debe8f856c10.

[13] Karniadakis et al. «Physics-Informed Machine Learning: Theory and Applications». In: *arXiv preprint* (2024). Available online. URL: https://arxiv.org/pdf/2403.15989.

[14] Willard et al. «Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems». In: *Engineering Applications of Machine Learning* (2022).

[15] Von Rueden et al. «Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems». In: *Machine Learning Research* (2023).