School of Electrical and Computer Engineering

Physics Informed Neural Networks with Focus on the Solution of Inverse Problems Arising in Vibrations of Rods

---

Author:
Kalliopi Nikolou

Committee:
Prof. M Zervakis (ECE),
Prof. G Stavroulakis (PEM),
Prof. M. Lagoudakis (ECE)

Chania, June 2025

Σχολή Ηλεκτρολόγων Μηχανικών και
Μηχανικών Υπολογιστών

Φυσικά Ενημερωμένα Νευρωνικά Δίκτυα για την Έμφαση στην Επίλυση Αντίστροφων Προβλημάτων στην Ταλάντωση Ραβδών

Φοιτήτρια: Καλλιόπη Νικολού

Επιτροπή:
Καθ. Μ. Ζερβάκης (ΗΜΜΥ),
Καθ. Γ. Σταυρουλάκης (ΜΠΔ),
Καθ. Μ. Λαγουδάκης (ΗΜΜΥ)

Χανιά, Ιούνιος 2025

Declaration of Authorship

I, Kalliopi Nikolou, declare that this thesis titled, “Physics Informed Neural Networks with Focus on the Solution of Inverse Problems Arising in Vibrations of Rods” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"*Sometimes, the bad things that happen in our lives put us directly on the path to the best things that will ever happen to us*"

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Abstract

Electrical and Computer Eng. Degree

Physics Informed Neural Networks with Focus on the Solution of Inverse Problems Arising in Vibrations of Rods

by Kalliopi Nikolou

Physics-Informed Neural Networks (PINNs) offer a transformative approach to solving complex problems governed by partial differential equations (PDEs). This thesis investigates the application of PINNs for both forward and inverse problems in the context of a vibrating rod system. The forward problem focuses on approximating the rod's displacement over time, given the governing PDE, initial conditions, and boundary constraints. The inverse problem centers on identifying unknown parameters, such as material properties, directly from observed data.

PINNs eliminate traditional numerical methods' reliance on mesh generation and efficiently handle challenges posed by high-dimensional systems, noisy data, and irregular geometries. By leveraging their ability to integrate physical laws with observational data, PINNs achieve accurate solutions while reducing computational complexity. This thesis further explores adaptive optimization techniques to improve convergence and accuracy, particularly for inverse problems where initial parameter estimates are far from their true values.

The results demonstrate that PINNs effectively model the physical behavior of the vibrating rod and accurately recover system parameters, showcasing their potential as a robust alternative to classical numerical methods. This work highlights the versatility and adaptability of PINNs, paving the way for future research into their application in more complex, real-world systems.

Περίληψη

Τα Φυσικά Ενημερωμένα Νευρωνικά Δίκτυα (PINNs) αποτελούν μια επαναστατική προσέγγιση για την επίλυση πολύπλοκων προβλημάτων που περιγράφονται από μερικές διαφορικές εξισώσεις. Η παρούσα εργασία διερευνά την εφαρμογή των PINNs τόσο σε ορθά ζητήματα όσο και σε αντίστροφα, στο πλαίσιο ενός συστήματος ράβδου που εμφανίζει χαρακτηριστικά ταλάντωσης. Στο ορθό πρόβλημα, επικεντρωνόμαστε στην προσέγγιση της μετατόπισης της ράβδου σε συνάρτηση με τον χρόνο και τη θέση της, δεδομένης της υποκείμενης διαφορικής εξίσωσης, των αρχικών συνθηκών και των οριακών περιορισμών. Στο αντίστροφο ζήτημα, κύριος στόχος είναι η εύρεση άγνωστων παραμέτρων, όπως οι ιδιότητες του υλικού.

Τα PINNs εξαλείφουν την εξάρτηση των παραδοσιακών αριθμητικών μεθόδων και αντιμετωπίζουν αποτελεσματικά τις προκλήσεις των πολυδιάστατων συστημάτων. Εκμεταλλευόμενα την ικανότητά τους να ενσωματώνουν φυσικούς νόμους συνδυαστικά με πειραματικές παρατηρήσεις, τα PINNs πετυχαίνουν ακριβείς λύσεις ενώ παράλληλα μειώνουν την υπολογιστική πολυπλοκότητα. Η εργασία αυτή εξετάζει επίσης προσαρμοστικές τεχνικές βελτιστοποίησης για τη βελτίωση της ακρίβειας, ιδίως σε αντίστροφα ζητήματα όπου οι αρχικές εκτιμήσεις των παραμέτρων απέχουν σημαντικά από τις πραγματικές τιμές.

Τα αποτελέσματα αποδεικνύουν ότι τα PINNs μοντελοποιούν αποτελεσματικά τη φυσική συμπεριφορά της ράβδου που εμφανίζει ταλαντώσεις και ανακτούν με ακρίβεια τις παραμέτρους του συστήματος, αναδεικνύοντας έτσι τη δυναμική τους. Αυτή η εργασία υπογραμμίζει την ευελιξία των PINNs, ανοίγοντας τον δρόμο για μελλοντική έρευνα στην εφαρμογή τους σε πιο πολύπλοκα συστήματα.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisors, for their invaluable guidance, encouragement, and support throughout the course of this thesis. Their expertise and insightful feedback were instrumental in shaping the direction and quality of this work.

I am also profoundly grateful to my family and friends for their unwavering support and encouragement, not only during the challenging moments of this research but throughout my entire academic journey. Their belief in me has been a constant source of motivation.

A special thank you goes to my colleagues and peers for their collaboration, stimulating discussions, and shared experiences, which enriched my perspective and contributed significantly to my personal and professional growth.

Lastly, I would like to acknowledge the inspiration drawn from the broader research community, whose groundbreaking work has laid the foundation for this study. This thesis is a reflection of collective efforts and shared knowledge, and I am humbled to contribute to this evolving field.

Contents

List of Figures

List of Tables

List of Abbreviations

PINN	**P**hysics-**I**nformed **N**eural **N**etwork
PDE	**P**artial **D**ifferential **E**quation
BC	**B**oundary **C**ondition
IC	**I**nitial **C**ondition
MSE	**M**ean **S**quared **E**rror
L-BFGS-B	**L**imited-**m**emory **B**royden–**F**letcher–**G**oldfarb–**S**hanno with **B**ox constraints

Dedicated to family...

Chapter 1

Introduction to Machine Learning

1.1 AI and Machine Learning

In recent years, Artificial Intelligence (AI) and Machine Learning (ML) have transformed various fields, from healthcare and finance to entertainment and transportation. These technologies enable computers to learn from data, identify patterns, and make decisions with minimal human intervention. AI represents the broader concept of machines being able to carry out tasks in a way that we consider "smart," whereas Machine Learning is a subset of AI that focuses on the development of algorithms that allow computers to learn from and make predictions based on data.

Neural Networks, a specific type of ML model, are designed to mimic the structure and function of the human brain. They consist of interconnected layers of "neurons" that process information in ways that can capture complex, non-linear relationships in data. This capability makes neural networks particularly powerful for tasks involving image recognition, language processing, and other applications requiring sophisticated pattern recognition.

The combination of AI, ML, and Neural Networks has led to groundbreaking advancements in technology. Applications such as self-driving cars, virtual assistants, medical diagnosis systems, and recommendation engines are becoming increasingly common, changing the way we interact with the world and access information. These technologies are shaping the future and opening new frontiers in areas previously thought to be uniquely human domains.

This project explores the foundations of Machine Learning, Artificial Intelligence, and Neural Networks, focusing on the mechanisms that allow these systems to learn from data, make predictions, and improve over time. By understanding the fundamental principles of these technologies, we gain insight into how modern AI systems are designed and how they can be applied to solve real-world problems.

1.2 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are computational models inspired by the human brain. They play a key role in machine learning, especially in tasks involving complex patterns and high-dimensional data. ANNs consist of layers of interconnected ”neurons” that process input data to produce an output, similar to how biological neurons operate.

Key Concepts in Artificial Neural Networks

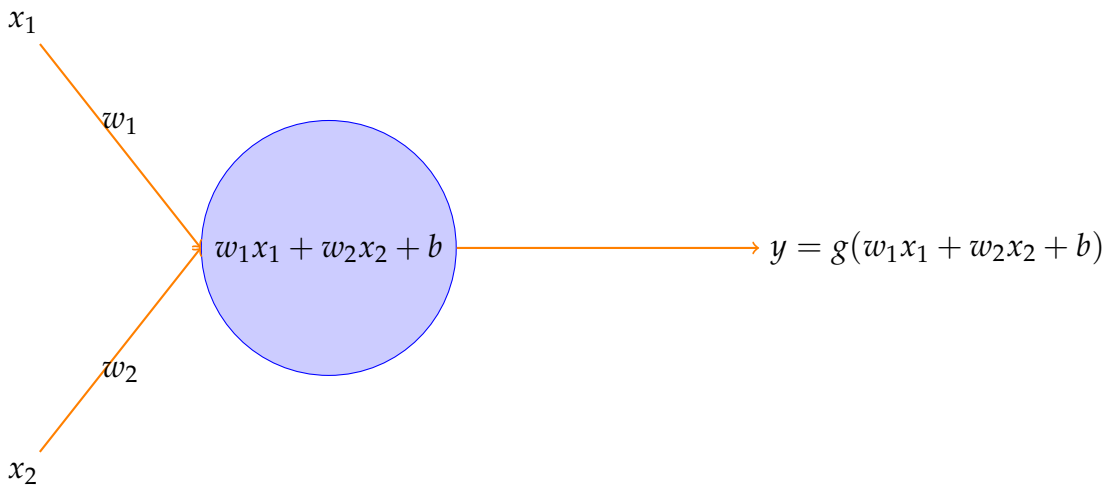


Figure 1.1: Illustration of a simple neuron structure highlighting weights and bias

1.2.1 Neurons

The basic units in an ANN, also known as nodes or units, are called neurons. Each neuron receives inputs, applies a weight to each input, sums them, applies an activation function, and produces an output. This process is inspired by biological neurons.

1.2.2 Layers

- **Input Layer**: This is where the network receives the initial data. The number of neurons in the input layer usually corresponds to the number of features in the dataset.
- **Hidden Layers**: These intermediate layers process the input data by applying weights and activation functions. ANNs can have one or many hidden layers,

and the model's complexity increases with more layers. If an ANN has many hidden layers, it is considered a *deep neural network* (DNN).

- **Output Layer**: The final layer that produces the network's prediction or output. For example, in a classification task, the output might be the probability of each class.

1.2.3 Weights and Biases - w_1, w_2, b

Each connection between neurons has an associated weight, determining the signal strength from one neuron to another. Biases are additional parameters that shift the activation function, helping the network to fit the data better.

1.2.4 Activation Functions - $g(x)$

Activation functions decide whether a neuron should be "activated" based on the weighted sum of its inputs. Common activation functions include:

- **Sigmoid**: Outputs values between 0 and 1, often used for binary classification tasks.

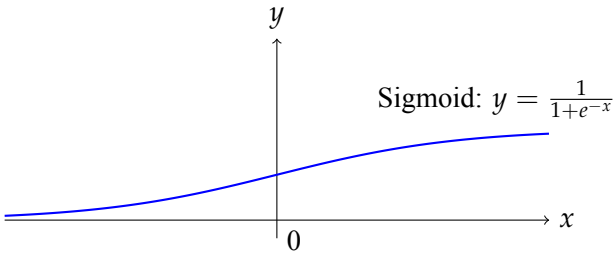


Figure 1.2: Sigmoid activation function: mapping real numbers to the [0,1] range

- **ReLU (Rectified Linear Unit)**: Outputs zero for negative inputs and the input itself for positive values, helping to address issues like vanishing gradients.

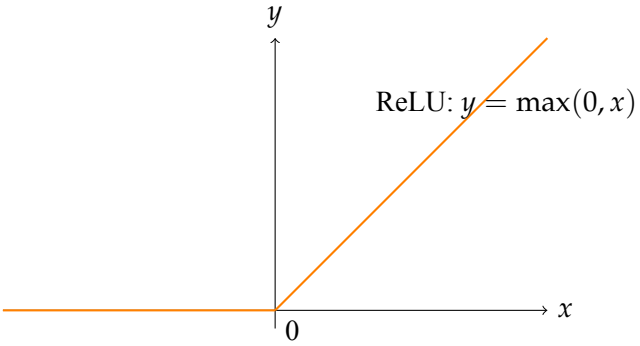


Figure 1.3: ReLU activation function: a popular non-linear function in deep learning

- **Tanh**: Similar to sigmoid but outputs values between -1 and 1, which can center the data.

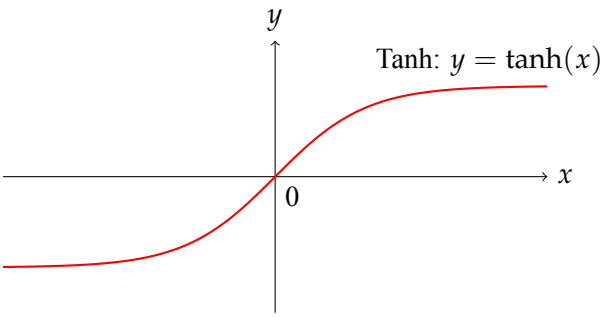


Figure 1.4: Tanh activation function: producing outputs from -1 to 1, centered at zero

1.2.5 Forward Propagation

Forward propagation is the process by which input data passes through the network layers, generating predictions. Each layer's output is calculated by taking the weighted sum of inputs, adding a bias, and applying an activation function.

1.2.6 Backward Propagation (Backpropagation)

Backward propagation is a training algorithm used to minimize the error by adjusting weights and biases. During backpropagation, the network calculates the error (difference between predicted output and actual output) and propagates it backward, updating weights to minimize it using techniques like gradient descent.

1.3 Mean Squared Error (MSE) and Gradient Descent Algorithms

1.3.1 Mean Squared Error (MSE)

Mean Squared Error (MSE) is a widely used loss function, particularly in regression tasks. MSE measures the average squared difference between predicted values $\hat{y}_i$ and actual values y_i. By squaring the differences, MSE penalizes larger errors more than smaller ones, which can be helpful when we want to avoid large deviations in our predictions [6].

The formula for MSE is as follows:

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

The goal during training is to minimize the MSE by adjusting the model's parameters. Lower MSE values indicate that the predictions are closer to the actual values, signaling better model performance.

1.3.2 Gradient Descent Algorithms

Gradient descent is an optimization technique used to minimize the loss function, such as MSE, by iteratively adjusting the model parameters. Several variants of gradient descent exist, each with unique features and applications. We will discuss some of the most widely used algorithms: Adam, RMSProp, and L-BFGS-B.

RMSProp (Root Mean Square Propagation)

RMSProp adapts the learning rate for each parameter by maintaining a moving average of the squared gradients, which stabilizes the updates and accelerates convergence. It is particularly effective for non-stationary objectives, making it suitable for many deep learning applications [35].

Adam (Adaptive Moment Estimation)

Adam combines ideas from Momentum and RMSProp. It keeps an exponentially decaying average of past gradients (momentum) and past squared gradients (adaptive learning rate) to stabilize and accelerate convergence [14].

L-BFGS-B (Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Box constraints)

L-BFGS-B is a quasi-Newton optimization method that approximates the second-order derivatives (Hessian matrix) of the loss function to optimize the parameters [19]. Unlike standard gradient descent, which only considers the gradient, L-BFGS-B uses approximations of the inverse Hessian matrix to provide more accurate updates.

1.4 Example: Neural Network Regression with Forward and Backward Propagation

This example demonstrates a simple neural network performing regression using one hidden layer and the sigmoid activation function. We will go through the forward pass, compute the Mean Squared Error (MSE), and then use backpropagation to update the weights.

1.4.1 Network Architecture

Consider a neural network with:

- 1 input neuron,
- 1 hidden layer with 2 neurons,
- 1 output neuron.

Let:

- x denote the input,
- $w_{1,1}, w_{1,2}$ be the weights from the input to the hidden layer neurons,
- $w_{2,1}, w_{2,2}$ be the weights from the hidden layer neurons to the output.

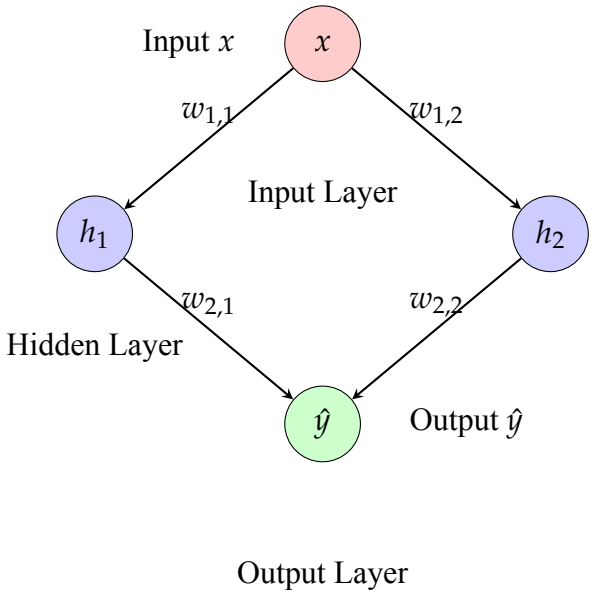


Figure 1.5: A simple feedforward neural network structure demonstrating forward pass

1.4.2 Forward Propagation

For an input x, the network computes the following steps:

Hidden Layer Activation

Each neuron in the hidden layer computes an activation as follows:

$$z_{1,1} = w_{1,1} \cdot x \quad \text{and} \quad z_{1,2} = w_{1,2} \cdot x$$

Using the sigmoid activation function:

$$a_{1,1} = \sigma(z_{1,1}) = \frac{1}{1 + e^{-z_{1,1}}}$$
$$a_{1,2} = \sigma(z_{1,2}) = \frac{1}{1 + e^{-z_{1,2}}}$$

where $a_{1,1}$ and $a_{1,2}$ are the activated outputs of the hidden neurons.

Output Layer Activation

The output neuron computes:

$$z_2 = w_{2,1} \cdot a_{1,1} + w_{2,2} \cdot a_{1,2}$$

Assuming a linear activation function for regression, the output $\hat{y}$ is:

$$\hat{y} = z_2$$

1.4.3 Loss Calculation

We use the Mean Squared Error (MSE) as the loss function:

$$L = \frac{1}{2}(y - \hat{y})^2$$

where y is the actual target value, and $\hat{y}$ is the predicted output.

1.4.4 Backpropagation

To minimize the loss, we need to compute the gradients of L with respect to the weights $w_{1,1}, w_{1,2}, w_{2,1}$, and $w_{2,2}$.

Gradient at the Output Layer

For the output layer, we calculate:

$$\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2}$$

Since $\hat{y} = z_2$, we have:

$$\frac{\partial L}{\partial z_2} = (\hat{y} - y)$$

Now, the gradients with respect to the weights $w_{2,1}$ and $w_{2,2}$ are:

$$\frac{\partial L}{\partial w_{2,1}} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_{2,1}} = (\hat{y} - y) \cdot a_{1,1}$$
$$\frac{\partial L}{\partial w_{2,2}} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_{2,2}} = (\hat{y} - y) \cdot a_{1,2}$$

Gradient at the Hidden Layer

For the hidden layer neurons, we need to compute:

$$\frac{\partial L}{\partial z_{1,1}} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_{1,1}} \cdot \frac{\partial a_{1,1}}{\partial z_{1,1}}$$

Using the sigmoid derivative $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, we get:

$$\frac{\partial a_{1,1}}{\partial z_{1,1}} = a_{1,1}(1 - a_{1,1})$$

Therefore,

$$\frac{\partial L}{\partial z_{1,1}} = (\hat{y} - y) \cdot w_{2,1} \cdot a_{1,1}(1 - a_{1,1})$$

Similarly,

$$\frac{\partial L}{\partial z_{1,2}} = (\hat{y} - y) \cdot w_{2,2} \cdot a_{1,2}(1 - a_{1,2})$$

Finally, the gradients with respect to $w_{1,1}$ and $w_{1,2}$ are:

$$\frac{\partial L}{\partial w_{1,1}} = \frac{\partial L}{\partial z_{1,1}} \cdot \frac{\partial z_{1,1}}{\partial w_{1,1}} = (\hat{y} - y) \cdot w_{2,1} \cdot a_{1,1}(1 - a_{1,1}) \cdot x$$

$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial z_{1,2}} \cdot \frac{\partial z_{1,2}}{\partial w_{1,2}} = (\hat{y} - y) \cdot w_{2,2} \cdot a_{1,2}(1 - a_{1,2}) \cdot x$$

1.4.5 Weight Update

Using gradient descent, we update each weight w with a learning rate α:

$$w_{1,1} = w_{1,1} - \alpha \frac{\partial L}{\partial w_{1,1}}$$

$$w_{1,2} = w_{1,2} - \alpha \frac{\partial L}{\partial w_{1,2}}$$

$$w_{2,1} = w_{2,1} - \alpha \frac{\partial L}{\partial w_{2,1}}$$

$$w_{2,2} = w_{2,2} - \alpha \frac{\partial L}{\partial w_{2,2}}$$

These updates are applied iteratively to minimize the loss function L and improve the model's predictions.

1.5 From small to deep neural networks

In recent years, deep neural networks (DNNs) have transformed fields ranging from computer vision and natural language processing to healthcare and finance. Deep neural networks are essentially artificial neural networks (ANNs) with multiple hidden layers. These layers allow DNNs to learn complex representations and capture intricate patterns in data, making them powerful tools for tackling complex tasks [17, 9]. This section explores why deep neural networks have become indispensable in modern machine learning and artificial intelligence.

1.6 Ability to Learn Hierarchical Representations

A fundamental factor behind the effectiveness of deep neural networks is their ability to learn complex hierarchical representations from data. In traditional machine learning, feature engineering is often required to transform raw data into meaningful features for a model to use. However, in DNNs, each layer builds upon the previous one, gradually learning higher-level, more abstract representations of the data [5].

- **Low-Level Features**: In the early layers of a DNN, the model learns low-level features, such as edges and textures in image data or basic word representations in text data.
- **Mid-Level Features**: Intermediate layers capture more complex patterns, such as shapes in images or phrases in text.
- **High-Level Features**: In deeper layers, the network learns high-level features that are more abstract, like objects in images or semantic meaning in text.

The hierarchical nature of deep neural networks enables them to handle raw data directly, reducing the need for feature engineering and allowing the network to automatically learn meaningful features for the task at hand [17].

1.6.1 High Capacity to Model Complex Patterns

The depth of a neural network, or the number of layers, significantly impacts its ability to model complex patterns. DNNs with many layers can represent highly non-linear functions, allowing them to capture complex relationships within data [9]. This is especially advantageous in tasks where traditional models, which are often limited to linear or shallow non-linear representations, may struggle.

- **Non-Linear Transformations**: Each layer in a DNN applies a non-linear transformation, enabling the network to learn non-linear relationships.
- **Universal Approximation**: According to the universal approximation theorem, neural networks can approximate any continuous function given sufficient depth and neurons. Deep networks can model intricate functions with fewer neurons per layer compared to shallow networks.

1.6.2 Scalability with Large Datasets

Deep neural networks thrive in data-rich environments. As the size of available datasets has grown exponentially, so too has the effectiveness of DNNs [8]. Unlike traditional machine learning models that may overfit or underperform with very large datasets, DNNs can scale effectively, learning more accurate representations as more data is introduced.

1.7 Versatility and Flexibility

Deep neural networks are highly versatile and can be adapted to various architectures to tackle different types of data and tasks. For example:

- **Convolutional Neural Networks (CNNs)**: Specialized for spatial data, CNNs excel in image processing tasks by leveraging convolutional layers to capture spatial hierarchies [15].
- **Recurrent Neural Networks (RNNs)**: Designed for sequential data, RNNs are commonly used in natural language processing and time-series analysis [9].

- **Transformer Networks**: Introduced for sequence-to-sequence tasks, transformers have become the state-of-the-art architecture for NLP, enabling models to capture long-range dependencies in text data [36].

1.7.1 Transfer Learning and Pretrained Models

One of the practical advantages of deep neural networks is the concept of transfer learning [26]. Transfer learning involves taking a pretrained model on a large dataset and fine-tuning it on a specific task or domain, which can drastically reduce the training time and improve performance on smaller datasets.

1.7.2 Robustness to Noise and Variability

Deep neural networks are more robust to noise and variability in data due to their ability to learn robust representations [9]. This robustness allows DNNs to generalize well even when the input data contains noise or variations.

1.7.3 Applications and Real-World Impact

The power of deep neural networks has led to their widespread adoption in various industries, significantly impacting modern society. Notable applications include computer vision, NLP, healthcare, and autonomous systems [17].

Chapter 2

Vibrating Rod Physics and Physical Informed Neural Networks (PINNs)

2.1 Introduction

In Chapter 1, the fields of application of artificial neural networks and their basic operating principles were thoroughly discussed. In this chapter, we will analyze how neural networks can address issues in physics and mathematics. Scientists constantly seek new and efficient ways to mathematically model the world for detailed study. For instance, the birth and death of a population are influenced by external factors. If one can mathematically correlate these phenomena, it becomes possible to make long-term predictions about the future of specific populations on various planets.

Furthermore, a nuclear physicist, understanding how the fuel of a nuclear reactor interacts with the surrounding temperature, can maintain the safe operation of the nuclear facility at desired levels. Many such scenarios are modeled using ordinary differential equations (ODEs). Differential equations arise in various mathematical and scientific contexts (both social and physical). Modern mathematical descriptions alter the use of differential notation.

Numerous mathematicians have studied differential equations and contributed to this field, including Isaac Newton, Gottfried Wilhelm Leibniz, the Bernoulli family, Riccati, Clairaut, Jean le Rond d'Alembert, and Leonhard Euler. An elementary example is Newton's second law, expressing the relationship between the displacement x and time t of an object with force F through the differential equation:

$$m\frac{\partial^2 x(t)}{\partial t^2} = F(x(t)) \tag{2.1}$$

This formulation characterizes the motion of a mass m, where F denotes the acting force and x is the position as a function of time t. A large number of physical processes can be described using partial differential equations (PDEs). A classic example is the Navier–Stokes system of equations [3], which stems from fundamental conservation principles—specifically, conservation of mass, momentum, and energy—and governs the behavior of fluids. By solving these equations within a specific domain and under appropriate initial and boundary constraints, one can simulate and predict fluid motion with precision.

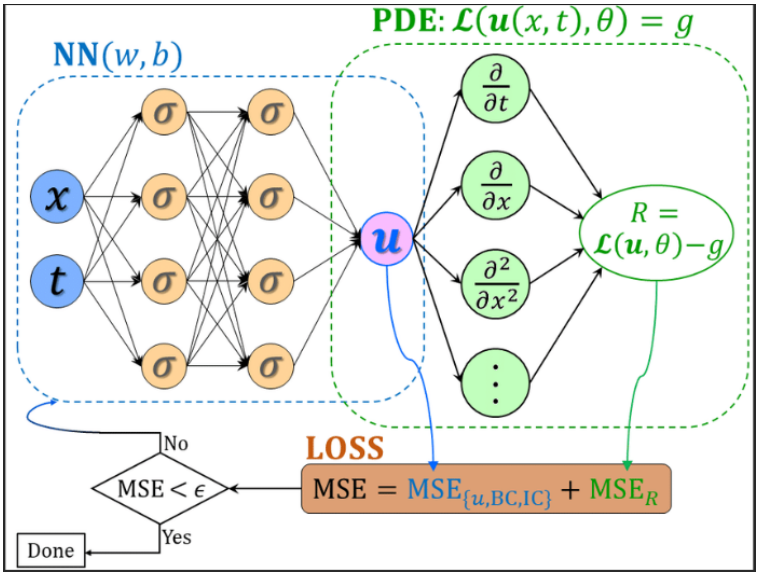
However, exact analytical solutions to these PDEs are rarely attainable in real-world cases, which makes numerical strategies such as the finite difference method, the finite element method, or the finite volume method indispensable. These computational techniques rely on prior assumptions, equation linearization, and accurate discretization across both temporal and spatial dimensions.

Recently, the intersection of deep learning and differential equation solving has given rise to a new research area within scientific machine learning (SciML). This paradigm leverages the powerful approximation properties of neural networks [11], which are capable of modeling high-dimensional functions when trained with sufficient data [1]. Nonetheless, conventional deep neural networks do not inherently incorporate the physical constraints of the underlying phenomena. Their predictive accuracy still hinges on the problem's geometry and the completeness of initial and boundary data. In the absence of such constraints, solutions produced by the network may be physically invalid or non-unique.

To overcome these limitations, Physics-Informed Neural Networks (PINNs) have been introduced. These networks embed physical laws directly into their training objectives, allowing them to learn not only from data but also from the differential equations that govern the system's dynamics. Consequently, PINNs can generalize well even when data is sparse or partially missing, provided that some information about the physical problem is available [2].

PINNs enable the solution of diverse problems in scientific computing and represent a significant advancement in the design of neural PDE solvers. Once trained, a PINN can be used to infer values over different computational grids or geometries without retraining [20]. Furthermore, they employ automatic differentiation (AD) [4] to compute derivatives involved in the governing equations—an efficient and modern alternative to symbolic or numerical differentiation that is particularly suited to neural network applications.

Figure 2.1: PINN



2.2 State of the art

In contemporary literature, the first comprehensive proposal for the use of Neural Networks (NNs) in solving differential equations, accompanied by implementation, is recognized as the work titled *Artificial Neural Networks for Solving Ordinary and Partial Differential Equations* [16]. This work, affiliated with the Department of Computer Science at the University of Ioannina, is considered crucial. It's worth noting that the initial idea of using neural networks for solving differential equations was proposed in the early 1990s through theoretical works, some of which are referred to in the bibliography. Examples include the work titled *Neural Algorithms for Solving Differential Equations* [18] and *The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks* [21]. The researchers mentioned above mainly came from scientific fields, particularly applied mathematics and computer science. The method was introduced to the field of mechanics, specifically in structural analysis, through the work titled *A Neural Network Approach to the Modelling, Calculation and Identification of Semi-Rigid Connections in Steel Structures* [34].

A characteristic work that presents the PINN) method in its current form is titled *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations* [29], where the principles of the method are defined, its programming implementation is analyzed, and it is applied to the partial differential equations Burger's Equation and Schrödinger Equation. This work serves as an excellent guide for someone starting to engage with the topic.

The study of the PINNs method has gained particular momentum from 2017 onwards. Besides the works mentioned above, several additional studies have been published by researchers worldwide. Recent works from the last year include *Physics-Informed Machine Learning* and *Physics-Informed Neural Networks for Elastic Plate Problems* [27], where the method is applied to solve the differential equation related to the Kirchhoff Plate Bending problem.

Notably, researchers such as Georgios E. Stavroulakis and Aliki D. Mouratidou from the Technical University of Crete have made significant contributions to the application of PINNs in computational mechanics. In their work *A Gentle Introduction to Physics-Informed Neural Networks, with Applications in Static Rod and Beam Problems* [23], they provide an accessible introduction to PINNs, applying them to static rod and beam problems, thus serving as a foundational resource for understanding the implementation of PINNs in structural mechanics.

Furthermore, in the study titled *Ensemble of Physics-Informed Neural Networks for Solving Plane Elasticity Problems with Examples* [24], Mouratidou and Stavroulakis explore the use of ensemble PINNs to solve plane elasticity problems, demonstrating the effectiveness of combining multiple neural networks to enhance solution accuracy in elasticity equations.

Additionally, their research presented at the 16th World Congress in Computational Mechanics, *Nonlinear Interaction in Composites Using Physics Informed Neural Networks* [25], investigates the integration of unilateral contact mechanics laws within the PINN framework, showcasing the potential of deep learning approaches in addressing complex contact mechanics problems in composite structures.

These contributions underscore the versatility and effectiveness of PINNs in addressing complex problems in computational mechanics, particularly in modeling interactions within composite materials and structural elements.

2.2.1 Modeling and Computation

A general nonlinear hyperbolic partial differential equation (PDE), such as the wave equation, can be written as:

$$u_{tt} + N[u;\lambda] = 0, \quad x \in \Omega, \quad t \in [0,T],$$

where $u(t,x)$ is the solution, $N[\,\cdot\,;\lambda]$ is a nonlinear operator parameterized by λ, and $\Omega \subset \mathbb{R}^D$ is the spatial domain. Hyperbolic PDEs describe wave-like phenomena

such as vibrations and oscillations in mechanical structures, including bars, strings, and membranes.

In the presence of measurement noise or incomplete knowledge of the physical model, Physics-Informed Neural Networks provide a powerful framework to address two main types of problems:

- Data-driven solutions
- Data-driven discovery

of hyperbolic partial differential equations.

Solving Data-Driven Hyperbolic Partial Differential Equations

The data-driven *PDE* solution ([29]) involves estimating the unknown state $u(t,x)$ based on measurements z, assuming the form of the PDE and its parameters λ are known:

$$u_{tt} + N[u] = 0, \quad x \in \Omega, \quad t \in [0,T],$$

We define the residual function $f(t,x)$ as:

$$f := u_{tt} + N[u],$$

and use a neural network to approximate $u(t,x)$, training it by minimizing the total loss function L_{tot}:

$$L_{tot} = L_u + L_f,$$

where $L_u = u - z$, with u being the network's output and z the real measurements. The term L_f represents the mean squared error of the residual, calculated by substituting the network output into the PDE.

This method enables the integration of physical laws and data for modeling hyperbolic systems, such as bar oscillations, in a computationally efficient manner ([30]).

Discovering Data-Driven Hyperbolic Partial Differential Equations

In the case of incomplete system state measurements z, the data-driven discovery of the *PDE* ([28]) aims to recover both the unknown state $u(t,x)$ and the underlying model parameters λ from the observed data. The PDE is expressed as:

$$u_{tt} + N[u;\lambda] = 0, \quad x \in \Omega, \quad t \in [0,T],$$

Defining:

$$f := u_{tt} + N[u;\lambda],$$

the neural network is trained to minimize:

$$L_{tot} = L_u + L_f,$$

where $L_u = u - z$ and L_f is again the mean squared residual error based on the governing hyperbolic PDE.

This approach allows for the discovery of dynamic models governed by hyperbolic PDEs, enabling the creation of fully differentiable surrogate models suitable for prediction, control, and interpretation of wave-like phenomena ([31, 22, 7]).

2.3 Formulation of Motion Equations through Newtonian Principles

In mechanical systems composed of discrete masses, motion equations can be constructed by applying Newton's second law individually to each mass in its respective direction of movement. The total number of such equations matches the number of degrees of freedom, as does the number of natural vibration modes.

Conversely, in continuous systems—such as rods or beams—mass and elasticity are distributed throughout the structure. These systems theoretically possess an infinite number of degrees of freedom. For example, a simply supported beam can exhibit an infinite array of vibration modes, each involving a different number of half-wave shapes. This raises an important question: does one need an infinite number of motion equations to describe such systems?

Surprisingly, the answer is no. In most practical scenarios involving bars, shafts, or slender beams, a single governing equation of motion suffices, despite the presence of infinitely many possible mode shapes. While some structures like arches or shells may require multiple equations, their number remains finite and manageable.

To derive such an equation for a continuous medium, one typically considers an infinitesimally small segment of the system and applies Newton's second law locally. This leads to a partial differential equation (PDE) that describes the system's dynamic behavior. Once this general equation is obtained, boundary conditions are applied to find its specific solutions.

Solving these boundary-specific equations often produces a characteristic equation involving complex mathematical functions, known as the frequency equation. In special cases, this equation can be solved analytically to yield exact expressions for the system's natural frequencies. More often, numerical methods are employed to estimate these values with sufficient accuracy.

This approach, commonly referred to as the "exact method," can yield highly accurate results for many canonical engineering problems. However, deriving the required equations can be analytically challenging or even infeasible for complex geometries. Therefore, approximate numerical techniques—though less precise—are often preferred for such cases.

Engineers and researchers using such approximations should validate them by comparing with known exact solutions in simpler systems. For instance, when analyzing a complex mechanical structure like a robotic arm with variable cross-sections, it's advisable to test the method on a simpler uniform beam under known loading, to ensure reliability.

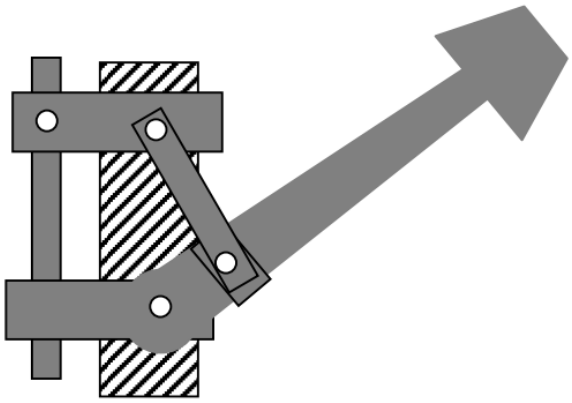


Figure 2.2: Schematic application of Newton's law to an infinitesimal segment (Source : [12])

Numerical simulations are typically suited for exploring the effects of varying parameters by running multiple input scenarios. However, the exact method provides deeper theoretical insight and allows for analytical validation. For this reason, we will begin by laying out the standard procedure for deriving equations of motion in continuous media.

Figure 2.3: Sample structural system: rod with loading (Source : [12])

Summary of Procedure for Motion Equation Derivation:

- **Step 1:** Identify a differential segment of the vibrating structure and draw the corresponding free-body diagram.
- **Step 2:** Express internal forces and moments in terms of the displacement variable, using:
  - (a) stress-resultant relations,
  - (b) constitutive laws (stress–strain),
  - (c) geometric relations (strain–displacement).
- **Step 3:** Combine all relevant force components in the direction of motion to compute the net internal action.
- **Step 4:** Include any applied dynamic forces, projecting them in the direction of motion.
- **Step 5:** Relate the total net force or moment to the mass times acceleration using Newton's second law, yielding the governing PDE.

2.3.1 Undamped Free Vibrations

In scenarios involving free vibration, there are no externally applied forces, and thus Step 4 of the general procedure becomes redundant. The governing differential equation is solved by incorporating the boundary conditions relevant to the specific problem. This process produces what is known as the characteristic or frequency equation, usually represented in determinant form. The roots of this equation correspond to the system's natural frequencies.

Furthermore, once these frequencies are known, the associated mode shapes (or natural modes) can be computed using the same general solution, constrained by the original boundary conditions. This concept will be made clearer through practical demonstrations in the next sections. Initially, we will focus on fundamental one-dimensional

structures—such as rods, shafts, and beams—before extending the analysis to more complex, two-dimensional frameworks. This chapter will strictly examine the calculation of natural frequencies and vibration modes; the analysis of motion resulting from specific initial conditions will be addressed in a later chapter.

2.3.2 Axial Vibrations in Uniform Bars

Motion Equation Development:

We begin by considering axial (longitudinal) vibration in a bar—one of the simplest examples of continuous vibration systems. Figure 2.4 illustrates a small segment of a bar with variable cross-sectional area $A(x)$, composed of a linear elastic material with density ρ and Young's modulus E. At time t, the longitudinal displacement at position x is given by $u(x,t)$, and the internal axial force by $F(x,t)$, assumed positive when tensile.

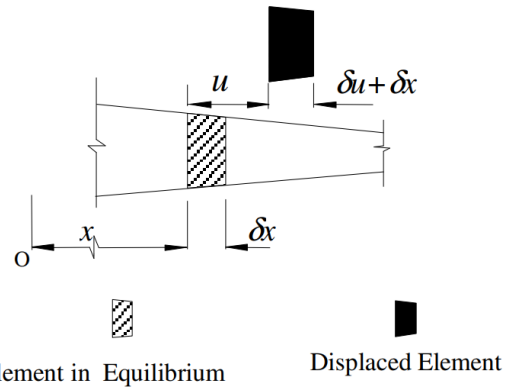


Figure 2.4: Displacement and force in a bar segment (Source : [12])

We now derive the equation of motion step-by-step:

Step 1: Consider a differential bar element of length δx, and draw its free-body diagram, as shown in Figure 2.5.

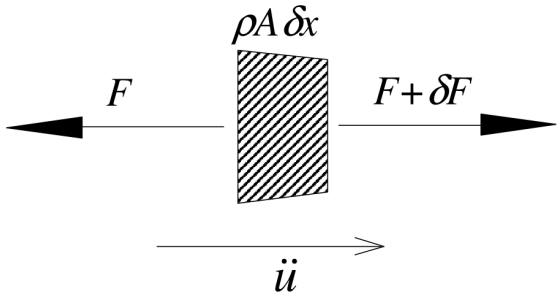


Figure 2.5: Free body diagram of a bar element (Source : [12])

Step 2: Express the internal force F in terms of displacement via physical laws:

- (a) Axial force and stress: $F = A\sigma$
- (b) Hooke's Law (stress-strain): $\sigma = E\varepsilon$
- (c) Strain and displacement: $\varepsilon = \frac{\partial u}{\partial x}$

Combining the above, we get:

$$F = EA\frac{\partial u}{\partial x}$$

The variation of force across the element is:

$$\delta F = \frac{\partial}{\partial x}\left(EA\frac{\partial u}{\partial x}\right)\delta x$$

Step 3: The net internal axial force on the element is simply δF, since opposing forces cancel.

Step 4: No external loads are acting, so δF is the total dynamic force.

Step 5: Apply Newton's second law:

$$\delta F = \rho A \delta x \cdot \frac{\partial^2 u}{\partial t^2}$$

This yields the following governing partial differential equation:

$$\frac{\partial}{\partial x}\left(EA\frac{\partial u}{\partial x}\right) = \rho A\frac{\partial^2 u}{\partial t^2}$$

In the case of a homogeneous bar with constant E and A, the equation simplifies to:

$$\frac{E}{\rho}\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2}$$

or, rearranged:

$$\frac{E}{\rho}\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = 0$$

2.4 The Role of Initial and Boundary Conditions in Differential Equations for Rod Analysis

In solving differential equations that describe the behavior of rods, initial and boundary conditions are fundamental. These conditions provide specific information about

the rod's placement, constraints, and external forces acting on it, all of which are essential to determine an accurate solution to the differential equation governing the system's dynamics. By defining initial and boundary conditions, we can simulate the real-world physical constraints that influence the behavior of a rod, whether it is fixed, free, or subjected to other configurations.

2.5 Importance of Initial and Boundary Conditions

In the context of rods, the primary differential equation often takes the form of the wave equation or Euler-Bernoulli beam equation, depending on whether we're modeling longitudinal vibrations or bending behavior. Boundary conditions specify the behavior of the rod at its endpoints, while initial conditions define its state at the beginning of observation (usually time $t = 0$).

- **Initial Conditions**: These define the initial displacement and/or velocity of the rod. For example, an initial condition could specify that the rod starts from rest or that it has an initial velocity along its length. This initial information sets the basis for how the rod will behave dynamically over time.
- **Boundary Conditions**: These define constraints at the endpoints of the rod and vary based on how the rod is fixed or allowed to move. Boundary conditions are essential for accurately simulating the physical setting of the rod, as they define the degrees of freedom at the endpoints. The nature of these conditions (fixed, free, or mixed) dramatically influences the solution to the differential equation and thus the rod's behavior.

2.6 Types of Boundary Conditions and Their Physical Interpretations

Boundary conditions are typically classified based on the nature of constraints at the rod's endpoints. These constraints are often dictated by physical setups such as fixed-fixed, free-free, and fixed-free conditions:

2.6.1 Fixed-Fixed Boundary Conditions

In a fixed-fixed configuration, both ends of the rod are anchored, preventing displacement and rotation. This type of boundary condition is common in structures where both ends are immobile, such as beams anchored in concrete supports. Mathematically, fixed-fixed conditions imply:

- **Displacement Boundary Condition**: $u(0,t) = 0$ and $u(L,t) = 0$, where $u(x,t)$ is the displacement along the length of the rod (from $x = 0$ to $x = L$) at any time t.

Fixed-fixed conditions create high constraints on the rod, resulting in specific vibration modes and frequencies that differ from other configurations. This setup restricts the rod's degrees of freedom, yielding higher natural frequencies and a distinct vibrational behavior.

2.6.2 Free-Free Boundary Conditions

In a free-free configuration, both ends of the rod are unconstrained, allowing for displacement and rotation without any external restriction. Such conditions apply to rods that are suspended in space or only loosely connected at the ends. Mathematically, free-free conditions can be expressed as:

- **Neumann Boundary Condition**: $\frac{\partial u}{\partial x}(0,t) = 0$ and $\frac{\partial u}{\partial x}(L,t) = 0$, indicating that there is no external force applied at the ends.

Under free-free conditions, the rod can move more freely, often resulting in lower natural frequencies compared to the fixed-fixed setup. The lack of external constraint at the ends allows for more complex vibrational modes, including rigid-body modes where the entire rod translates or rotates as a unit without deformation.

2.6.3 Fixed-Free Boundary Conditions

In the fixed-free configuration, one end of the rod is anchored (fixed), while the other is free to move or rotate. This setup is frequently seen in cantilever beams, such as diving boards or projecting structures. Mathematically, these conditions are:

- **Fixed Boundary Condition at One End**: $u(0,t) = 0$, restricting displacement at the anchored end.
- **Free Boundary Condition at the Other End**: $\frac{\partial u}{\partial x}(L,t) = 0$, allowing the free end to move without an external force constraint.

This boundary condition creates an asymmetric response in the rod. The fixed end resists movement, while the free end has maximal displacement, leading to a unique set of natural frequencies and vibrational modes. The fixed-free setup typically exhibits a combination of bending and longitudinal motion, depending on the applied force and initial conditions.

2.7 How Initial and Boundary Conditions Define Vibrational Modes

The initial and boundary conditions collectively influence the vibrational modes, resonant frequencies, and dynamic behavior of the rod.

- **Fixed-Fixed Rod**: Constrains both endpoints, leading to symmetric vibration modes where nodes (points of zero displacement) appear at both ends. The vibration frequencies are generally higher, and the rod behaves similarly to a taut string in fixed musical instruments.
- **Free-Free Rod**: Lacks constraints at either endpoint, allowing the rod to vibrate as a whole without fixed nodes at the ends. This setup introduces rigid-body modes in addition to standard vibrational modes, leading to a more complex vibrational spectrum.
- **Fixed-Free Rod**: With one end fixed and the other free, the vibrational modes are asymmetric. The free end typically experiences higher amplitude vibrations than the fixed end, making this configuration sensitive to external forces or initial disturbances.

2.8 Solutions of vibrating rod based on its ends

2.8.1 Fixed - Fixed Ends

Given the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \tag{2.2}$$

where c is the wave speed.

We'll use the method of separation of variables to solve this partial differential equation.

Let $u(x,t) = X(x)T(t)$, then the wave equation becomes:

$$X(x)T''(t) = c^2 X''(x)T(t) \tag{2.3}$$

Dividing both sides by $c^2 X(x)T(t)$, we get:

$$\frac{1}{c^2}\frac{T''(t)}{T(t)} = \frac{X''(x)}{X(x)} = -\lambda \tag{2.4}$$

where λ is a separation constant.

Solving the Spatial Part ($X(x)$):

The general solution to the spatial part is:

$$X(x) = C\cos(\sqrt{\lambda}x) + D\sin(\sqrt{\lambda}x) \tag{2.5}$$

Applying the boundary condition $u(0,t) = u(1,t) = 0$:

When $x = 0$, $X(0) = C\cos(0) + D\sin(0) = C$.

Since we have the condition $u(0,t) = 0$, this implies that $X(0)T(t) = 0$, which means that $C = 0$ because otherwise the solution wouldn't satisfy the boundary condition.

So, by setting $C = 0$, we ensure that the solution $u(x,t)$ satisfies the boundary condition $u(0,t) = 0$.

$$X(1) = D\sin(\sqrt{\lambda}\cdot 1) = 0 \implies D\sin(\sqrt{\lambda}) = 0 \tag{2.6}$$

This gives two cases:

Solving the Spatial Part ($X(x)$):

This gives two cases:

1. $D = 0$, which leads to the trivial solution.
2. $\sin(\sqrt{\lambda}) = 0$, which implies $\sqrt{\lambda} = n\pi$, where n is a positive integer.

So, $\lambda = (n\pi)^2$.

Therefore, the spatial part of the solution becomes:

$$X_n(x) = D_n\sin(n\pi x) \tag{2.7}$$

where $n = 1, 2, 3, \ldots$ and D_n are constants.

Solving the Temporal Part ($T(t)$):

The general solution to the temporal part is:

$$T(t) = A\cos(ct\sqrt{\lambda}) + B\sin(ct\sqrt{\lambda}) \tag{2.8}$$

Now, let's analyze the coefficients A and B using the initial conditions.

Initial Condition Analysis:

1. **Initial Condition 1:** $u(x,0) = A_0\sin(\phi\pi x)$
2. **Initial Condition 2:** $\frac{\partial u}{\partial t}(x,0) = 0$

Analysis of Coefficients A and B:

- **Condition 1:** To satisfy the first initial condition, B cannot be zero.
- **Condition 2:** To satisfy the second initial condition, we need to ensure that the derivative of $u(x,0)$ with respect to t does not contain any sine term. Hence, B must be zero.

Therefore, $B = 0$ to satisfy both initial conditions.

Now, we have:

$$T(t) = A\cos(ct\sqrt{\lambda}) \tag{2.9}$$

Determination of A:

Given the first initial condition, when $t = 0$:

$$T(0) = A\cos(0) = A$$

Comparing with $u(x,0) = A_0\sin(\phi\pi x)$, we have $A = 1$.

Final Solution:

Therefore, the complete solution $u(x,t)$ is given by:

$$u(x,t) = A_0\sin(\phi\pi x)\cdot\cos(ct\phi\pi) \tag{2.10}$$

where A_0 is the initial amplitude, ϕ is a constant, c is the wave speed, and $\lambda = (n\pi)^2$ with n being a positive integer.

2.8.2 Free - Free ends

Analysis of Initial Conditions:

1. **Initial Condition 1:** $u(x,0) = A \cdot \cos(\phi\pi x)$
2. **Initial Condition 2:** $\frac{\partial u}{\partial t}(x,0) = 0$
3. **Boundary Conditions:** $\frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(1,t) = 0$

Analysis of Coefficients A and B:

- **Condition 1:** To satisfy the first initial condition, B cannot be zero.
- **Condition 2:** To satisfy the second initial condition, we need to ensure that the derivative of $u(x,0)$ with respect to t does not contain any sine term. Hence, B must be zero.
- **Condition 3:** To satisfy the boundary conditions, D must be zero to ensure that the spatial part of the solution satisfies the boundary conditions at $x = 0$ and $x = 1$.

Therefore, $B = 0$ to satisfy both initial conditions and the spatial boundary conditions.

Now, we have:

$$T(t) = A\cos(ct\sqrt{\lambda})$$

Determination of A:

Given the first initial condition, when $t = 0$:

$$T(0) = A\cos(0) = A$$

Comparing with $u(x,0) = A \cdot \cos(\phi\pi x)$, we have $A = 1$.

Final Solution:

Therefore, the complete solution $u(x,t)$ is given by:

$$u(x,t) = A_0 \cos(\phi\pi x) \cdot \cos(ct\phi\pi)$$

where c is the wave speed, ϕ is a constant, and $\lambda = (n\pi)^2$ with n being a positive integer.

Chapter 3

Problem Statement

The aim of this chapter is to analyze in detail the objective of the thesis. In the previous chapters, we have examined how the differential equations of the vibrating rod are derived mathematically and found corresponding solutions based on the state of the rod's edges (e.g., free-free, fixed-free).

This thesis aims to leverage Physics-Informed Neural Networks (PINNs) in detail and demonstrate their potential as a powerful tool for solving physics problems with complex initial and boundary conditions, such as the vibrating rod problem.

A reasonable question that arises is why we need PINNs when we already know the solutions to the differential equations. The following points illustrate the advantages of using PINNs:

- **High-Dimensional Problems**: Traditional numerical methods (e.g., finite element or finite difference methods) struggle with high-dimensional problems due to the curse of dimensionality. PINNs, on the other hand, can efficiently solve high-dimensional PDEs because they leverage neural networks, which can represent complex functions in high-dimensional spaces.
- **Integration of Observational Data and Physics**: PINNs can seamlessly integrate observational data with known physics. In situations where there are partial measurements or noisy data, PINNs can optimize a solution that respects both the governing equations and the data, creating a more accurate model of reality.
- **Solving Inverse Problems**: PINNs are powerful for solving inverse problems, where the objective is to identify unknown parameters or functions in the differential equations. Traditional methods often require iterative approaches and can be computationally intensive, while PINNs can incorporate this search directly into the training process.

- **Adaptability to Different Types of Differential Equations**: PINNs can be adapted to handle different types of differential equations (e.g., ordinary differential equations (ODEs), partial differential equations (PDEs), stochastic equations) without needing significant changes to their underlying structure, unlike many traditional numerical methods.
- **Mesh-Free Nature**: Traditional numerical methods often require a mesh or grid, which can be computationally expensive and complex to implement for irregular domains. PINNs are mesh-free, meaning they can approximate solutions without discretizing the domain, making them highly adaptable to complex geometries.
- **Handling Singularities and Sharp Changes**: Problems with sharp changes or singularities can be challenging for standard numerical solvers. PINNs can learn these features more naturally if the neural network is trained appropriately, as they can represent complex non-linear behaviors.
- **Combining Physics with Empirical Data**: In scenarios where the physics is not fully understood or there is a combination of known physical laws and unknown empirical relationships, PINNs can blend known physics with learned behaviors from data, helping to bridge the gap.

Hence, this thesis serves as a proof-of-concept that PINNs can effectively solve both forward and inverse problems.

Given a training set $\mathcal{T}r$ consisting of observations from the vibration of a rod, represented as:

$$\mathcal{T}r = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\},$$

and considering the governing partial differential equation (PDE) for the vibrating rod:

$$\frac{\partial^2 u}{\partial t^2} = \lambda \frac{\partial^2 u}{\partial x^2},$$

where $u(x,t)$ is the displacement of the rod at position x and time t, and λ incorporates the rod properties:

$$\lambda = \sqrt{\frac{E}{\rho}},$$

with E denoting the Young's modulus and ρ the density of the rod, our objective is to search for the mapping function $(x,t) \mapsto u(x,t)$ that satisfies this PDE and the associated initial and boundary conditions.

3.1 Forward Problem Formulation

In the forward problem (**solve the diff. equation**), the total loss function $L(\theta;\tau)$ is built from multiple terms to ensure the model respects the physical laws and boundary conditions. This allows to approximate the solution $\hat{u}(x,t)$ by minimizing discrepancies with respect to both the differential equation and the specified initial and boundary conditions.

The loss components for the forward problem are defined as follows:

1. **Physics-Driven Loss**:

$$L_{\text{int}}(\theta;\tau_{\text{int}}) = \frac{1}{|\tau_{\text{int}}|} \sum_{(x_i,t_i)\in\tau_{\text{int}}} \left| \frac{\partial^2 \hat{u}}{\partial t^2}(x_i,t_i) - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}(x_i,t_i) \right|^2,$$

   where τ_{int} denotes the collocation points within the domain. This termforces the neural network to adhere to the PDE everywhere inside the domain by penalizing any deviation from the differential equation.

2. **Boundary Condition Loss at** $x = 0$:

$$L_{x=0}(\theta;\tau_{x=0}) = \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{u}(x_i,t_i)|^2,$$

   where $\tau_{x=0}$ denotes the set of boundary points at $x = 0$. This term enforces the boundary condition at the left end of the domain.

3. **Boundary Condition Loss at** $x = L$:

$$L_{x=L}(\theta;\tau_{x=L}) = \frac{1}{|\tau_{x=L}|} \sum_{(x_i,t_i)\in\tau_{x=L}} |\hat{u}(x_i,t_i)|^2,$$

   where $\tau_{x=L}$ is the set of boundary points at $x = L$. This term enforces the boundary condition at the right end of the domain.

4. **Initial Condition Loss for Displacement**:

$$L_{t=0}(\theta;\tau_{t=0}) = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{u}(x_i,t_i) - f(x_i)|^2,$$

   where $\tau_{t=0}$ represents the set of initial condition points at $t = 0$, and $f(x)$ is the initial displacement function. This term ensures that the solution matches the initial displacement condition.

5. **Initial Condition Loss for Velocity**:

$$L_{t=0}^{\text{par}}(\theta;\tau_{t=0}) = \frac{1}{|\tau_{t=0}|}\sum_{(x_i,t_i)\in\tau_{t=0}}\left|\frac{\partial\hat{u}}{\partial t}(x_i,t_i)\right|^2.$$

The total loss function for the forward problem combines all these components:

$$L(\theta;\tau) = L_{\text{int}}(\theta;\tau_{\text{int}}) + L_{x=0}(\theta;\tau_{x=0}) + L_{x=L}(\theta;\tau_{x=L}) + L_{t=0}(\theta;\tau_{t=0}) + L_{t=0}^{\text{par}}(\theta;\tau_{t=0}),$$

ensuring that the PINN respects the PDE, initial conditions, and boundary conditions across the entire domain.

Minimization Goal:

The target of the training process is to find the optimal network parameters θ^* by minimizing the total loss:

$$\theta^* = \arg\min_{\theta} L(\theta;\tau).$$

This minimization ensures that the solution to the forward problem while adhering to the physical constraints and observed conditions.

3.2 Inverse Problem Formulation

The aim of the inverse problem is to find the rod properties λ. In the context of solving the inverse problem, the total loss function is defined as the sum of multiple components that ensure the network respects **both the observed data and the governing physical laws**. The total loss $\mathcal{L}_{\text{total}}$ for the inverse problem is given by:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{x=0} + \mathcal{L}_{t=0}^{(1)} + \mathcal{L}_{x=L} + \mathcal{L}_{t=0}^{(2)} + \mathcal{L}_{\text{train}},$$

where:

1. **Physics-Driven Loss**:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{|\mathcal{T}_{\text{int}}|}\sum_{(x_i,t_i)\in\mathcal{T}_{\text{int}}}\left|\frac{\partial^2 u_\theta}{\partial t^2}(x_i,t_i) - \lambda\frac{\partial^2 u_\theta}{\partial x^2}(x_i,t_i)\right|^2.$$

2. **Boundary Condition Loss at** $x = 0$:

$$\mathcal{L}_{x=0} = \frac{1}{|\mathcal{T}_{x=0}|} \sum_{(x_i,t_i)\in\mathcal{T}_{x=0}} |u_\theta(x_i,t_i)|^2 .$$

3. **Initial Condition Loss for Displacement**:

$$\mathcal{L}^{(1)}_{t=0} = \frac{1}{|\mathcal{T}_{t=0}|} \sum_{(x_i,t_i)\in\mathcal{T}_{t=0}} |u_\theta(x_i,t_i) - f(x_i)|^2 .$$

4. **Boundary Condition Loss at** $x = L$:

$$\mathcal{L}_{x=L} = \frac{1}{|\mathcal{T}_{x=L}|} \sum_{(x_i,t_i)\in\mathcal{T}_{x=L}} |u_\theta(x_i,t_i)|^2 .$$

5. **Initial Condition Loss for Velocity**:

$$\mathcal{L}^{(2)}_{t=0} = \frac{1}{|\mathcal{T}_{t=0}|} \sum_{(x_i,t_i)\in\mathcal{T}_{t=0}} \left|\frac{\partial u_\theta}{\partial t}(x_i,t_i)\right|^2 .$$

6. **Data-Driven Loss**:

$$\mathcal{L}_{\text{train}} = \frac{1}{|\mathcal{T}_{\text{train}}|} \sum_{(x_i,t_i)\in\mathcal{T}_{\text{train}}} |u_\theta(x_i,t_i) - u_{\text{obs}}(x_i,t_i)|^2 .$$

$$\theta^*, \lambda^* = \arg\min_{\theta,\lambda} \mathcal{L}_{\text{total}}(\theta,\lambda).$$

The total loss function $\mathcal{L}_{\text{total}}$ ensures that the solution aligns with the governing PDE, satisfies the boundary and initial conditions, and fits the observed data, enabling effective parameter estimation for the inverse problem.

Chapter 4

Experiments

4.1 PINN Algorithm for solving forward problem

4.1.1 Vibration Equation with Dirichlet Conditions

We examine a classical one-dimensional wave propagation scenario, defined over a spatial domain with fixed boundary conditions of the Dirichlet type. The mathematical formulation is as follows:

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} = 4\dfrac{\partial^2 u}{\partial x^2}, & x \in [0,1],\ t \in (0,2) \\ u(0,t) = 0, & t \in (0,2] \\ u(1,t) = 0, & t \in (0,2] \\ u(x,0) = \sin(\pi x), & x \in (0,1) \\ \dfrac{\partial u}{\partial t}(x,0) = 0, & x \in (0,1) \end{cases} \tag{4.1}$$

This setup describes wave motion in a medium where both endpoints are held fixed. An analytical solution to this problem is known and can be expressed as $u(x,t) = \sin(\pi x)\cos(2\pi t)$. In the sections that follow, we will demonstrate how a Physics-Informed Neural Network can be used to approximate this solution numerically. The experimental design draws inspiration from the methodology proposed in [32].

Network Architecture: In our PINN-based approach to approximate $u(x,t)$, the neural network is constructed with two input neurons corresponding to the variables x and t, and a single output neuron providing the predicted value of u. The hidden layers consist of four fully connected layers, each composed of 50 neurons. This architecture was selected to balance approximation capability with computational efficiency.

PINN : Here, we consider a deep feedforward neural network (DeepPDE) whose main goal is to approximate some function, in our case $y(x,t)$ for any input (x,t).

This model is called feedforward because information flows through the function being evaluated from (x,t), through the intermediate computations, and finally to the output $u(x,t) \approx \hat{u}(x,t)$. There are no feedback connections in which outputs of the model are fed back into itself.

The architecture of the neural network used in this study is based on an input consisting of $d+1$ variables, represented as $\mathbf{x} = (x,t) \in \mathbb{R}^{d+1}$, and a single scalar output denoted by $\hat{u}(\mathbf{x};\theta)$. For this problem, we consider $d = 1$.

The function $\hat{u}(\mathbf{x};\theta)$, which serves as the neural network's approximation of the target solution, follows the structure detailed in [**6**], and is defined as follows:

$$\begin{aligned}
&\text{Input layer:} \quad \mathcal{N}^{(0)}(\mathbf{x}) = \mathbf{x} = (x,t) \in \mathbb{R}^2, \\
&\text{Hidden layers:} \quad \mathcal{N}^{(\ell)}(\mathbf{x}) = \sigma\left(W^{(\ell)}\mathcal{N}^{(\ell-1)}(\mathbf{x}) + b^{(\ell)}\right), \quad \ell = 1,2,\ldots,L=4, \\
&\text{Output layer:} \quad \hat{u}(\mathbf{x};\theta) = \mathcal{N}^{(L)}(\mathbf{x}) = W^{(L)}\mathcal{N}^{(L-1)}(\mathbf{x}) + b^{(L)} \in \mathbb{R}
\end{aligned} \tag{4.2}$$

Here, the notation includes:

- $\mathcal{N}^{(\ell)} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ is the output of the ℓ-th layer, consisting of N_ℓ neurons,
- $W^{(\ell)} \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $b^{(\ell)} \in \mathbb{R}^{N_\ell}$ denote the weight matrices and bias vectors respectively, with all parameters collectively denoted by $\theta = \{W^{(\ell)}, b^{(\ell)}\}_{\ell=1}^{L}$,
- σ represents the activation function applied element-wise; in this case, the hyperbolic tangent is employed, i.e., $\sigma(s) = \tanh(s)$ for $s \in \mathbb{R}$.

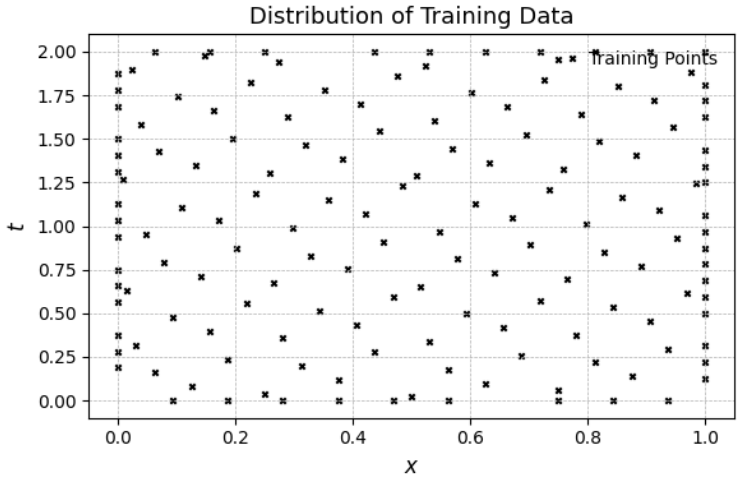


Figure 4.1: Training set of forward problem - Dirichlet BC

Training dataset. The general training set τ of this model is selected in the interior domain $\tau_{\text{int}} \subset (0,1) \times (0,2)$ and on the boundaries $\tau_{x=0} \subset \{0\} \times [0,2]$, $\tau_{x=1} \subset \{1\} \times [0,2]$, $\tau_{t=0} \subset (0,1) \times \{0\}$. Thus $\tau = \tau_{\text{int}} \cup \tau_{x=0} \cup \tau_{x=1} \cup \tau_{t=0}$.

The training set we used consisted of 200 samples $\{(x_i, t_j); u(x_i, t_j)\}_{i,j=1}^{200}$ where $u(x_k, t_k)$ is the solution of (1) at (x_k, t_k). 200 training samples were chosen from $(0,1) \times (0,2)$ and the rest was taken from the boundary of the domain. The plot 4.1 shows the training samples (x, t).

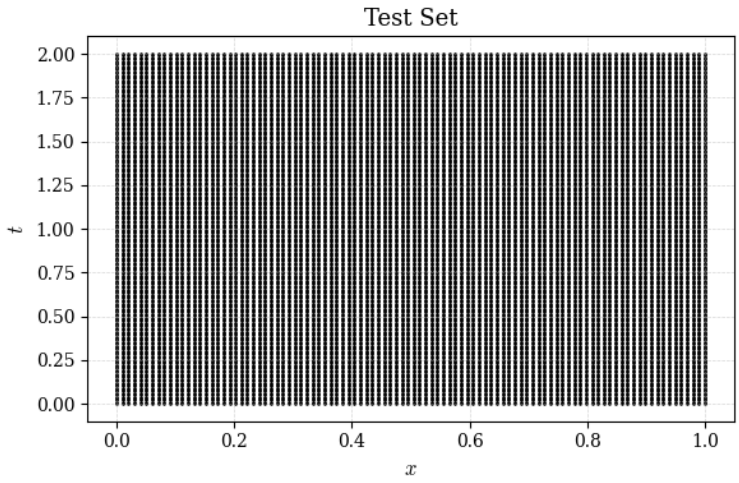


Figure 4.2: Test set of forward problem - Dirichlet BC

Test Set The test set 4.2 covers the whole domain area ($100 \times 200 = 20{,}000$ points).

Loss function During training, our loss is defined as the sum of the squared pointwise difference of each of the following equations:

$$L_{\text{int}}(\theta; \tau_{\text{int}}) = \frac{1}{|\tau_{\text{int}}|} \sum_{(x_i,t_i)\in\tau_{\text{int}}} \left| \frac{\partial^2 \hat{u}}{\partial t^2}(x_i, t_i) - 4\frac{\partial^2 \hat{u}}{\partial x^2}(x_i, t_i) \right|^2, \tag{4.3}$$

$$L_{x=0}(\theta; \tau_{x=0}) = \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{u}(x_i, t_i)|^2, \tag{4.4}$$

$$L_{x=1}(\theta; \tau_{x=1}) = \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{u}(x_i, t_i)|^2, \tag{4.5}$$

$$L_{t=0}(\theta; \tau_{t=0}) = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{u}(x_i, t_i) - \sin(\pi x_i)|^2, \tag{4.6}$$

$$L^*_{t=0}(\theta; \tau_{t=0}) = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} \left| \frac{\partial \hat{u}}{\partial t}(x_i, t_i) \right|^2. \tag{4.7}$$

$$L(\theta;\tau) = L_{\text{int}}(\theta;\tau_{\text{int}}) + L_{x=0}(\theta;\tau_{x=0}) + L_{x=1}(\theta;\tau_{x=1}) + L_{t=0}(\theta;\tau_{t=0}) + L^{*}_{t=0}(\theta;\tau_{t=0}) \tag{4.8}$$

Summing u, the training parameters are the following:

Parameter	Value
Optimizer	L–BFGS–B
Training steps	12000
Learning Rate	0.001
Activation Function	tanh
Hidden layers	4
Nodes per layer	50

4.1.2 Results of Forward Problem - Dirichlet BC

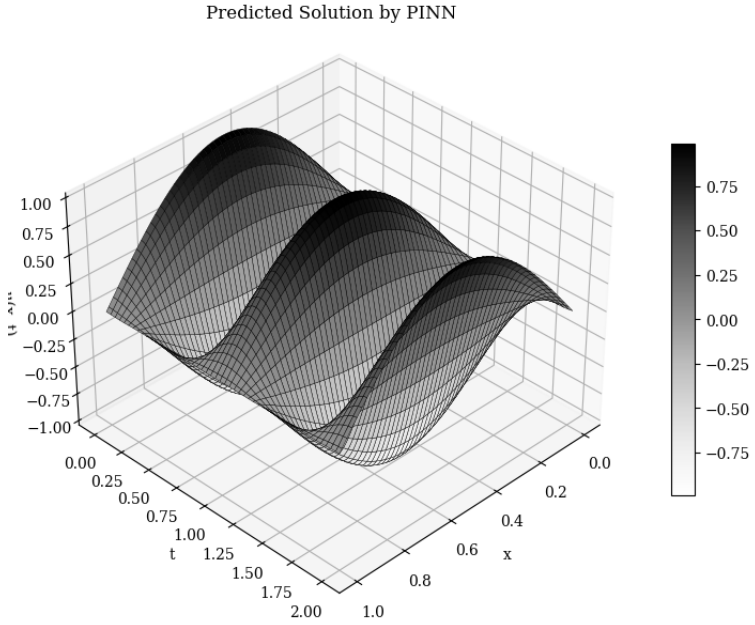


Figure 4.3: Prediction of forward problem - Dirichlet BC

The analysis of the results from training the forward problem using PINN reveals several significant insights into the model's predictive capabilities. **The achieved average mean squared error (MSE) of 0.001** indicates a high level of accuracy in the PINN's approximation of the true solution. However, an in-depth examination of the results provides a clearer understanding of the model's performance.

A key aspect to consider is the comparison between the estimated solution by the PINN (referred to as the "PINN state") and the exact solution (the "Exact state").

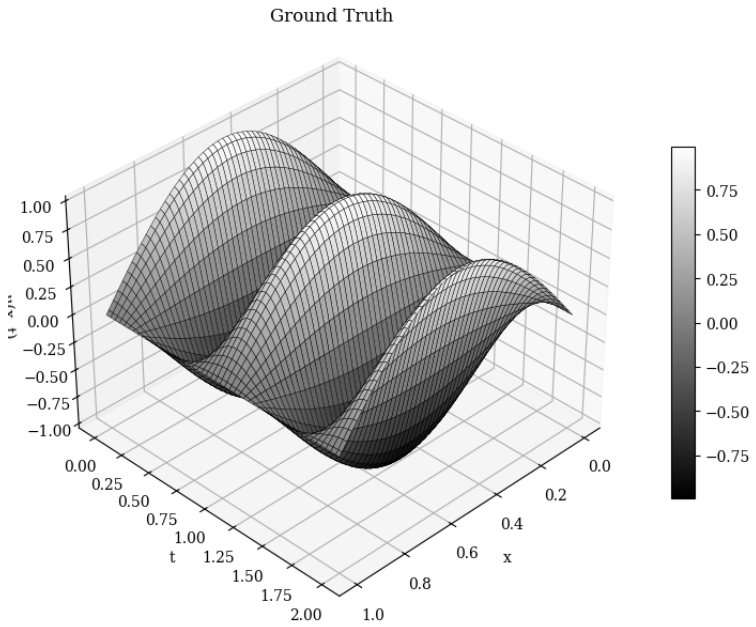


Figure 4.4: Ground Truth of forward problem - Dirichlet BC

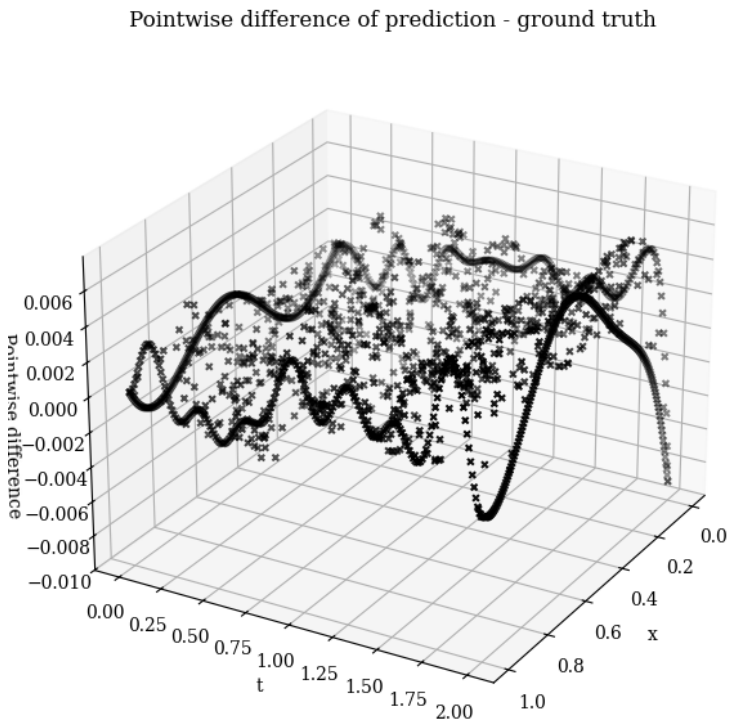


Figure 4.5: Pointwise difference between PINN state and exact state of forward problem - Dirichlet BC

The pointwise difference between these two, as depicted in the pointwise difference plot (Figure 4.5), shows a range from approximately -0.008 to 0.004, which is not statistically important error. In the context of this work, the choice to present the

pointwise difference, defined as

$$u_{\text{exact}} - u_{\text{pred}},$$

instead of the absolute or relative error, was made deliberately to retain both the **magnitude** and the **direction** of the error.

While metrics such as the *absolute error*

$$|u_{\text{exact}} - u_{\text{pred}}|$$

or the *relative error*

$$\frac{|u_{\text{exact}} - u_{\text{pred}}|}{|u_{\text{exact}}|}$$

are useful for quantifying the overall accuracy and are commonly employed in statistical analysis, they obscure whether the model is systematically **underestimating** or **overestimating** the true values.

The PINN state visualization (Figure 4.3) suggests that the model has captured the fundamental behavior of the problem, showcasing a wave-like structure that aligns well with the expected physical behavior.

The training configuration, which involved four hidden layers with 50 nodes each and the tanh activation function, appears to have supported the model's learning process effectively. The use of the L-BFGS-B optimizer over 12,000 training steps likely contributed to the smooth convergence of the loss function, enabling the model to reach a low average MSE. The choice of tanh as the activation function, known for its smooth and bounded nature.

In conclusion, the results demonstrate that the PINN framework, configured with suitable hyperparameters and training strategies, effectively approximates the forward problem governed by partial differential equations.

4.2 PINN Algorithm for solving inverse problem

Physics-informed neural networks offer an effective approach to solve inverse PDE problems, such as parameter identification in different types of PDEs. This capability motivates the use of PINN to discover λ that satisfies eq:inverse$_p$*roblem*) :

$$\begin{cases} \frac{\partial^2 u}{\partial t^2}(x,t) = \lambda \frac{\partial^2 u}{\partial x^2}(x,t), & 0 < x < 1,\ 0 < t < 2 \\ u(0,t) = 0, & 0 \le t \le 2 \\ u(1,t) = 0, & 0 \le t \le 2 \\ u(x,0) = \sin(\pi x), & 0 < x < 1 \\ \frac{\partial u}{\partial t}(x,0) = 0, & 0 < x < 1 \end{cases} \tag{4.9}$$

We utilized a dataset consisting of 1800 total training samples. Out of these, 1000 were sampled uniformly from the interior of the rectangular domain $(0,1) \times (0,2)$. Additionally, we allocated 200 points along each of the domain boundaries located at $x = 0$, $x = 1$, $t = 0$, and $t = 2$, with uniform spacing. Denote the interior training subset as $\mathcal{T}_{\text{in}}$, and the collective boundary training subset as $\mathcal{T}_{\partial\Omega} = \mathcal{T}_{x=0} \cup \mathcal{T}_{x=1} \cup \mathcal{T}_{t=0} \cup \mathcal{T}_{t=2}$.

An initial value of λ was assigned as 1 in our training process. The neural network framework is structured upon this foundation. We trained the model for a total of 6000 iterations (epochs), during which the weights are continuously updated. Let the neural network's output corresponding to input (x_i, t_i) be expressed as $\hat{u}(x_i, t_i)$.

The architecture of the neural network remains consistent with those adopted in subsequent sections. The optimization process is carried out using the Adam algorithm, which adjusts the network's parameters in accordance with the current value of λ. Furthermore, the value of λ is dynamically updated during training, with its optimal value being determined alongside the model parameters to minimize the prescribed loss function, as defined below:

$$L_{PDE} = \frac{1}{|\tau_{\text{int}}|} \sum_{(x_i,t_i)\in\tau_{\text{int}}} \left|\hat{u}_{tt}(x_i,t_i) - \lambda \hat{u}_{xx}(x_i,t_i)\right|^2, \tag{4.10}$$

$$L_{x=0} = \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} \left|\hat{u}(x_i,t_i)\right|^2, \tag{4.11}$$

$$L_{t=0_1} = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} \left|\frac{\partial \hat{u}}{\partial t}(x_i,t_i)\right|^2. \tag{4.12}$$

$$L_{x=1} = \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} \left|\hat{u}(x_i,t_i)\right|^2, \tag{4.13}$$

$$L_{t=0_2} = \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{u}(x_i,t_i) - \sin(\pi x_i)|^2, \tag{4.14}$$

$$L_{\text{data}} = \frac{1}{|\tau_{\text{int}}|} \sum_{(x_i,t_i)\in\tau_{\text{int}}} |\hat{u}(x_i,t_i) - u(x_i,t_i)|^2. \tag{4.15}$$

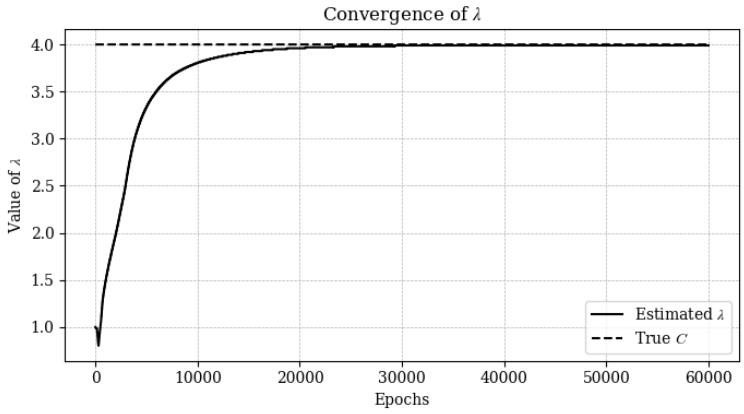


Figure 4.6: λ estimation - inverse problem

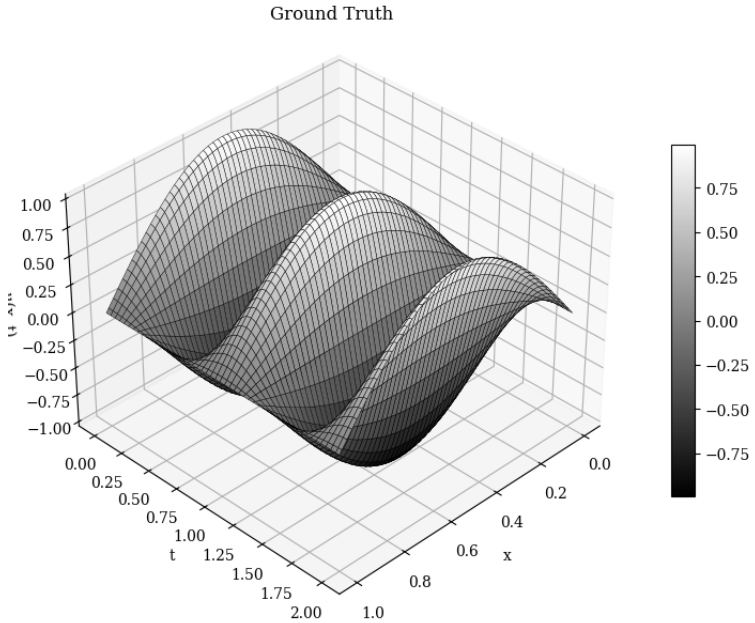


Figure 4.7: Ground truth of inverse problem

This initial value of λ, although small, enables the λ predictions from the model to nearly reach its true value after 30000 epochs. Even though we chose a very small initial value of λ, the model was successful in predicting the true value of λ after exactly 30000 epochs, as illustrated in Figure 4.6. The solution of the inverse problem generates the best found value of λ to be 3.99. We aim to see how this new model

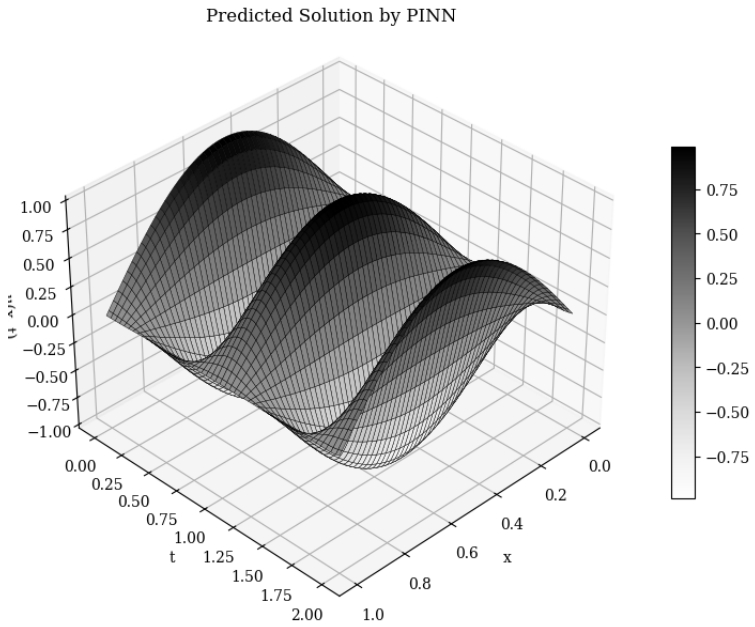


Figure 4.8: Prediction of inverse problem

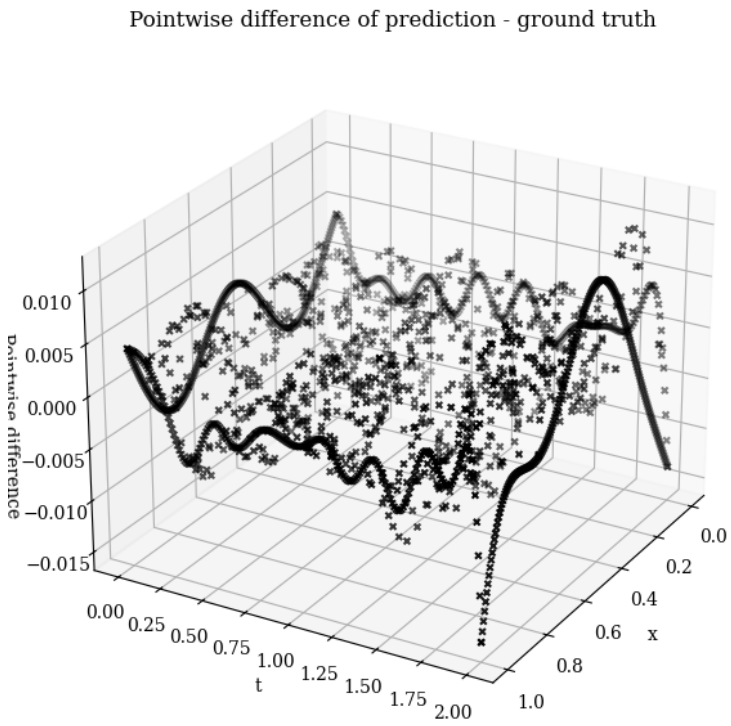


Figure 4.9: Pointwise difference between PINN prediction and ground truth of inverse problem

with this λ predicts the input data and compare it to the exact solution of our wave equation in 1. To better interpret the accuracy of the model let us observe the pointwise difference between these two states: Figure 4.9 suggests that the performance

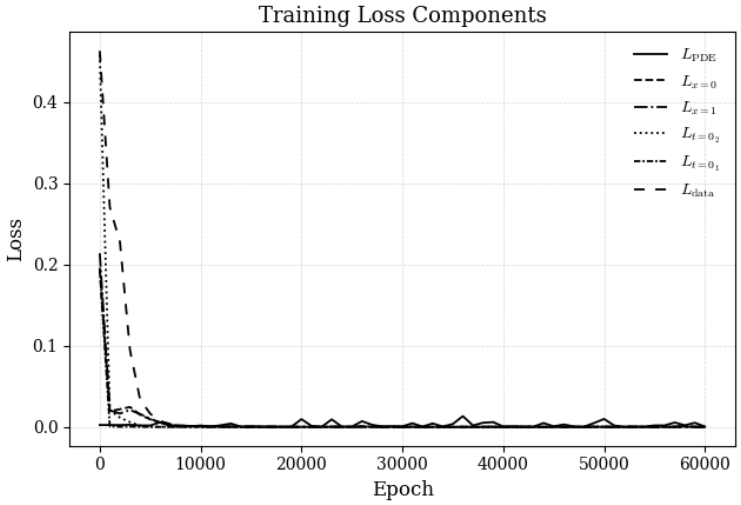


Figure 4.10: Components of loss functions - inverse problem

of our model is acceptable, by showing that the pointwise difference lay in the interval [-0.015, 0.010] units. We conclude that our developed machine learning model performs well with a negligible pointwise difference between the exact and PINNs predicted states.

4.3 PINN Algorithm for solving inverse problem - Adaptive Technique

In this section, we aim to improve the performance of our model by giving different weight to each component of loss function. From figure 4.10 is obtained that the L_{data} converges smoothly to zero, something that may be beneficial to estimate λ with the minimum error. Hence, the new loss function is :

$$
\begin{aligned}
L = 1 \times \frac{1}{|\tau_{\text{int}}|} \sum_{(x_i,t_i)\in\tau_{\text{int}}} \left| \frac{\partial^2 \hat{u}}{\partial t^2}(x_i,t_i) - \lambda \frac{\partial^2 \hat{u}}{\partial x^2}(x_i,t_i) \right|^2 \\
+ 1 \times \frac{1}{|\tau_{x=0}|} \sum_{(x_i,t_i)\in\tau_{x=0}} |\hat{u}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} \left| \frac{\partial \hat{u}}{\partial t}(x_i,t_i) \right|^2 \\
+ 1 \times \frac{1}{|\tau_{x=1}|} \sum_{(x_i,t_i)\in\tau_{x=1}} |\hat{u}(x_i,t_i)|^2 + 1 \times \frac{1}{|\tau_{t=0}|} \sum_{(x_i,t_i)\in\tau_{t=0}} |\hat{u}(x_i,t_i) - \sin(\pi x_i)|^2 \\
+ 7 \times \frac{1}{|\tau_{\text{int}}|} \sum_{(x_i,t_i)\in\tau_{\text{int}}} |\hat{u}(x_i,t_i) - u(x_i,t_i)|^2 .
\end{aligned}
\tag{4.16}
$$

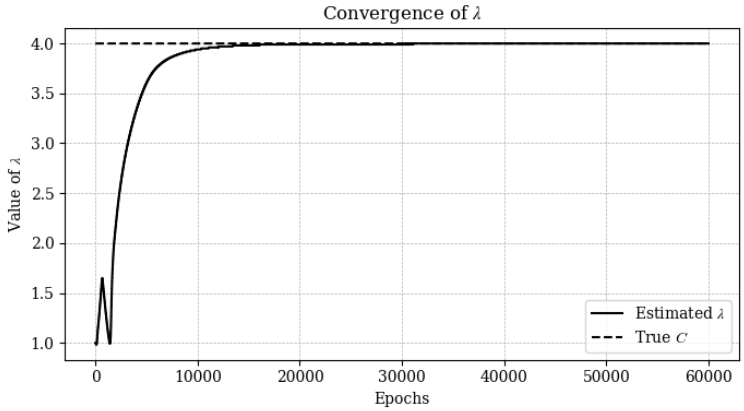


Figure 4.11: Adaptive technique - λ estimation - inverse problem

Our PINN model finds the best value of λ to be 4.0. Figure 4.11 shows that the predicted values of λ nearly reach its true value after 20000 epochs. Moreover, when using equal weights in the loss function as in the first example, we obtained the best predicted value of λ to be 3.99, whereas this model can reach a perfect prediction of λ equal to its true value. Let us know analyze the performance of this developed model by comparing its output of the input data (x, t) to the true value of $u(x, t)$.

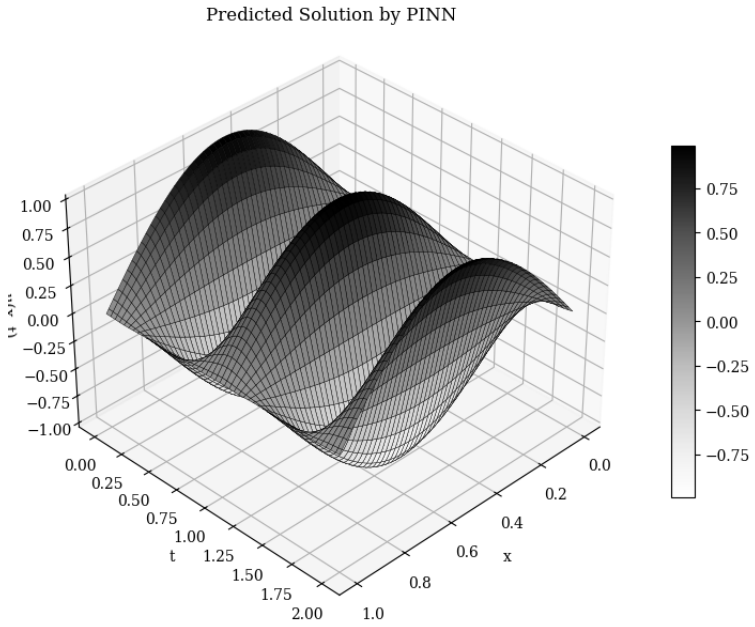


Figure 4.12: Adaptive technique - Prediction of inverse problem

Figure 4.14 illustrates that the discrepancy between the true solution and the output generated by the PINN remains within the interval $[-0.015, 0.005]$. Only a minimal portion of the domain reaches the extremes of this range. Based on this observation, we infer that the chosen weight configuration enhanced the model's accuracy in prediction.

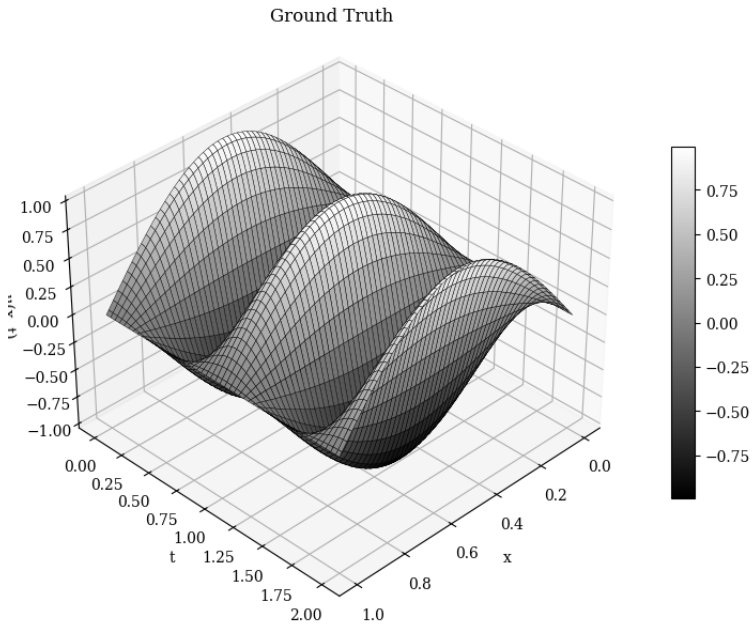


Figure 4.13: Adaptive technique - Ground Truth of inverse problem

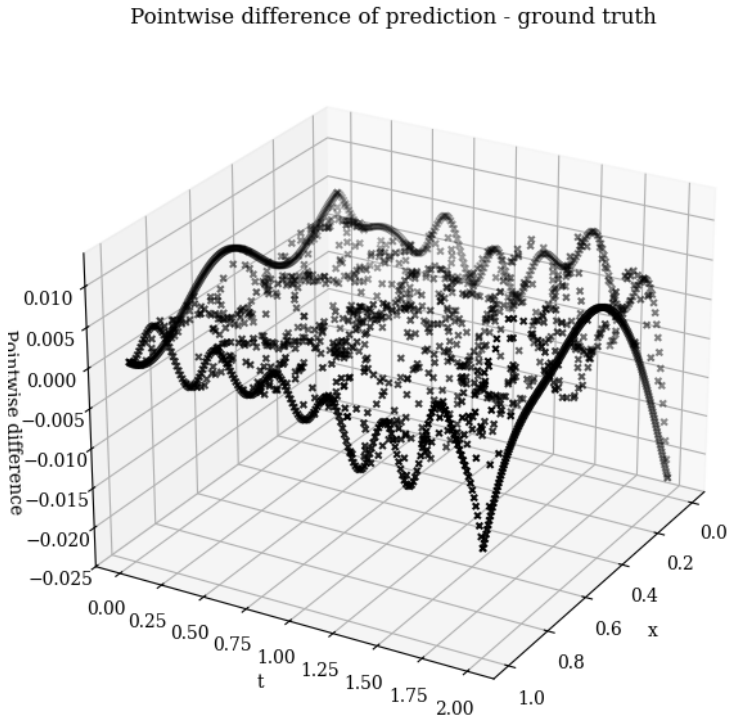


Figure 4.14: Adaptive technique - Pointwise difference between PINN prediction and ground truth inverse problem

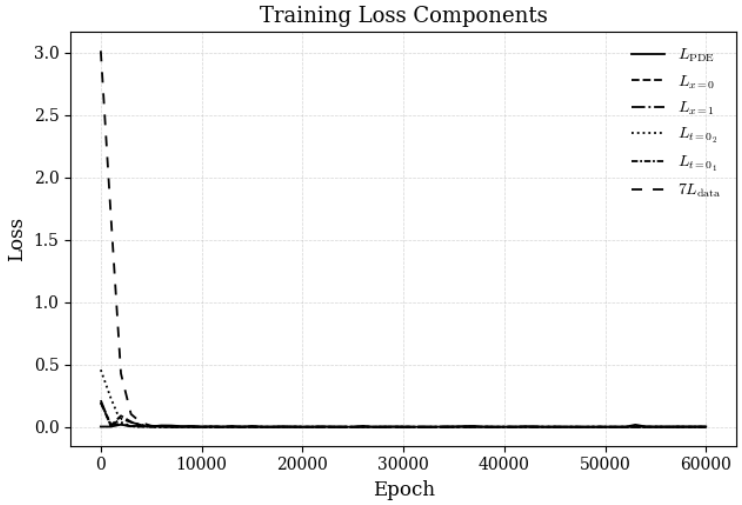


Figure 4.15: Adaptive technique - Components of loss functions - inverse problem

4.4 PINN Algorithm for solving inverse problem - λ far away from real value

We finally aim that our Machine Learning model performs well on solving the inverse problem, even though the initial value of λ is far from its true value. We integrate the following loss function in our model:

$$\begin{aligned} L = & \frac{1}{|T_{int}|} \sum_{(x_i,t_i)\in T_{int}} \left| \frac{\partial^2 u}{\partial t^2}(x_i,t_i) - \lambda \frac{\partial^2 u}{\partial x^2}(x_i,t_i) \right|^2 \\ & + \frac{1}{|T_{x=0}|} \sum_{(x_i,t_i)\in T_{x=0}} |u(x_i,t_i)|^2 + \frac{1}{|T_{x=1}|} \sum_{(x_i,t_i)\in T_{x=1}} |u(x_i,t_i)|^2 \\ & + \frac{1}{|T_{t=0}|} \sum_{(x_i,t_i)\in T_{t=0}} |u(x_i,t_i) - \sin(\pi x_i)|^2 \\ & + \frac{1}{|T_{int}|} \sum_{(x_i,t_i)\in T_{int}} |u(x_i,t_i) - y(x_i,t_i)|^2 \\ & + \frac{1}{|T_{t=0}|} \sum_{(x_i,t_i)\in T_{t=0}} \left| \frac{\partial \hat{u}}{\partial t}(x_i,t_i) \right|^2 \end{aligned} \tag{4.17}$$

The results of the experiments conducted on the PINN algorithm for solving the inverse problem with λ significantly far from its true value reveal critical insights into the model's performance and limitations. The primary objective of the study was to assess the ability of the PINN framework to estimate the parameter λ, which governs

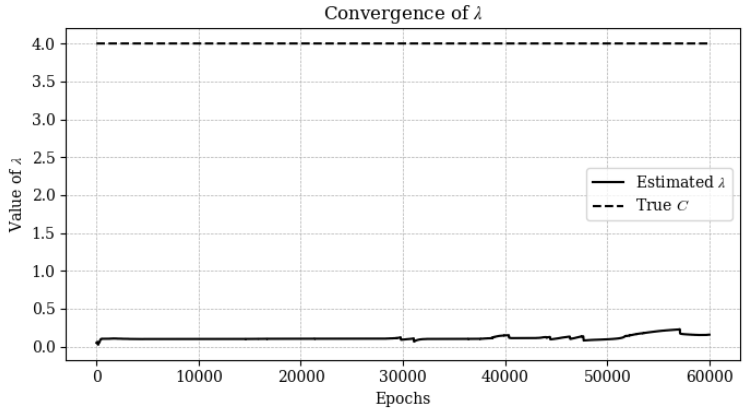


Figure 4.16: λ estimation - inverse problem λ far away from real value.

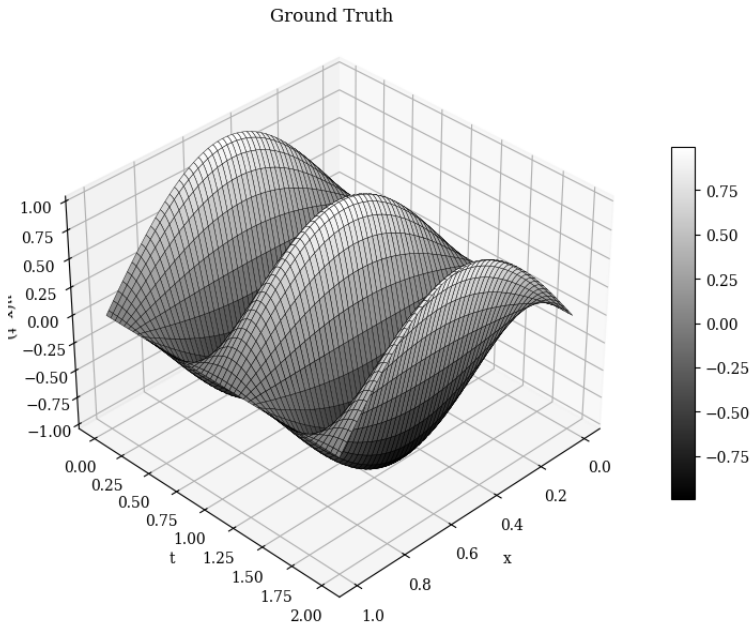


Figure 4.17: Ground Truth of inverse problem - λ far away from real value.

the dynamics of the underlying partial differential equation, and to recover the exact state of the system when the initial estimate of λ was set to 0.05, while the true value was 4.0. This setup presents a significant challenge, as it requires the model to adapt and converge effectively under highly unfavorable initial conditions.

Figure 4.16 illustrates the progression of the predicted value of λ throughout the training process. The results indicate that while the model makes some progress in adjusting λ from its initial value, it fails to converge to the true value of 4.0. The slow rate of improvement and the eventual plateauing of the predicted value suggest that the optimization process faces significant challenges. These difficulties could stem from a poorly conditioned loss landscape, or an inappropriate weighting of the loss

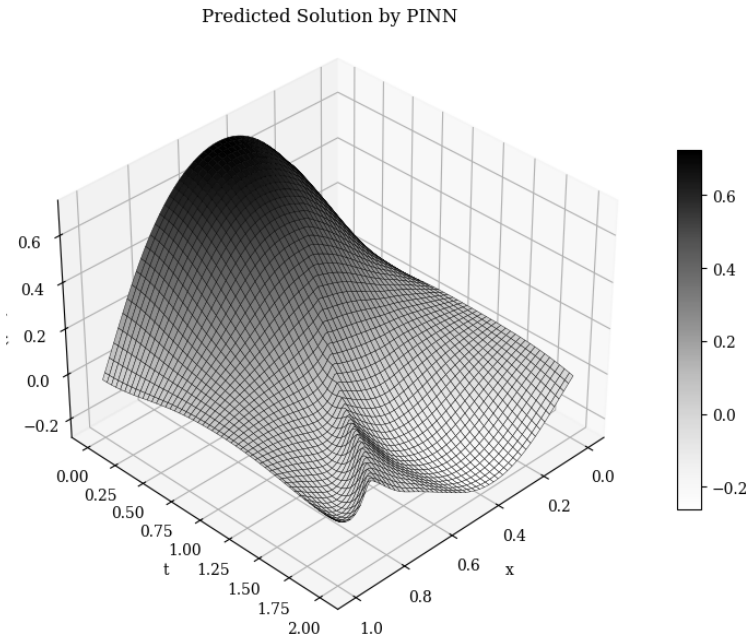


Figure 4.18: Prediction of inverse problem - λ far away from real value.

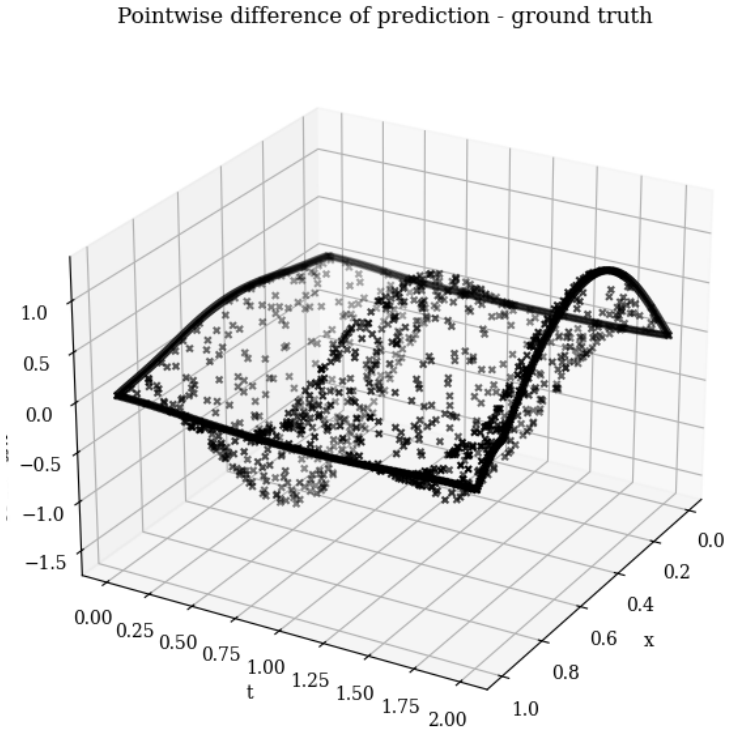


Figure 4.19: Pointwise difference between PINN prediction and ground truth of inverse problem - λ far away from real value.

components. The disparity between the predicted and true values of λ underscores the inherent sensitivity of the optimization process in inverse problems, especially when initial parameter guesses are far from the true values.

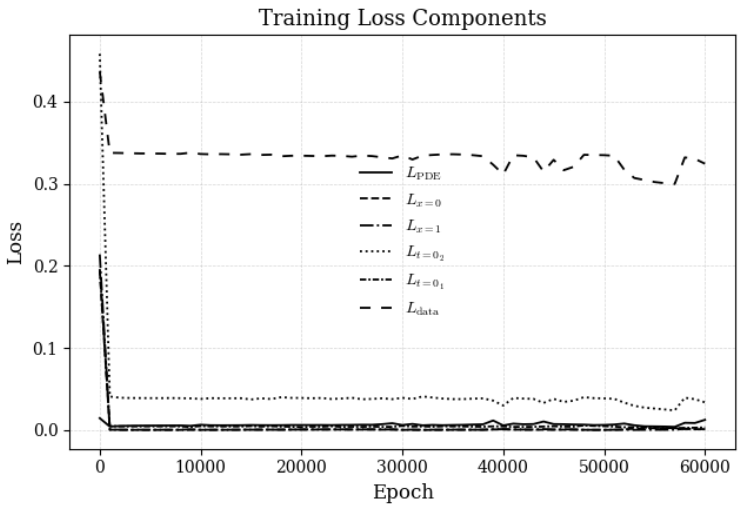


Figure 4.20: Components of loss functions - inverse problem - λ far away from real value.

The comparison between the exact state (Figure 4.18) and the PINN-predicted state provides further insight into the model's performance. Qualitatively, the PINN captures some aspects of the underlying dynamics of the system, as the overall structure of the predicted solution resembles the true solution. However, quantitative discrepancies are evident, particularly in regions with sharp gradients or oscillatory behavior. This mismatch can be attributed to the incorrect estimation of λ, which directly influences the governing PDE and propagates errors throughout the solution. The inability of the PINN to recover the exact state with high fidelity highlights the limitations of the framework in handling inverse problems with large parameter mismatches.

The pointwise difference between the exact state and the PINN-predicted state, as shown in Figure 4.19, further highlights the model's limitations. Errors are localized in specific regions of the domain, likely corresponding to areas of high curvature or nonlinearity in the solution. This observation aligns with theoretical expectations, as inaccuracies in λ estimation amplify in regions where the solution is most sensitive to parameter changes. The propagation of these errors underscores the importance of accurate parameter recovery in ensuring the overall reliability of the predicted solution.

The breakdown of loss function components over training epochs, presented in Figure 4.20, reveals additional insights into the optimization dynamics. Loss terms associated with boundary and initial conditions decrease rapidly during the early stages of training, indicating that the PINN effectively satisfies these constraints. However, the loss term tied to the PDE residual converges more slowly and remains relatively

high, reflecting the challenges in enforcing the physics constraints when λ is misestimated. This imbalance among the loss components suggests that the optimization process might benefit from a dynamic weighting strategy, which could reallocate focus to the most critical terms during different stages of training. Additionally, the stagnation of the PDE residual loss indicates potential difficulties with the gradient flow, which could be alleviated through advanced techniques such as learning rate scheduling, adaptive optimizers, or regularization methods designed specifically for PINNs.

The experiments reveal several challenges inherent in using PINNs for inverse problems with poor initial parameter estimates. The optimization process is particularly sensitive to the initial guess for λ, as the significant gap between the initial and true values exacerbates the difficulty of convergence. While the loss function is carefully designed to incorporate multiple components representing the physics, boundary, and initial conditions, the imbalance among these components appears to hinder effective optimization. Furthermore, the inability of the PINN to fully recover λ highlights the importance of developing strategies to improve parameter estimation in such settings. Techniques such as adaptive loss balancing, pretraining on similar problems, or leveraging domain knowledge could enhance the model's ability to converge to the correct solution.

In conclusion, the results demonstrate that while the PINN framework shows promise in capturing the qualitative features of the exact state, its ability to recover parameters and accurately predict the solution is limited when initial parameter estimates are far from the true values. The findings underscore the need for improved optimization strategies, refined loss formulations, and advanced techniques to guide the learning process.

4.5 PINN Algorithm for solving inverse problem - λ far away from real value - Adaptive algorithm

The adaptive method employed for solving the inverse problem with the PINN algorithm represents a significant improvement in tackling the challenges posed by the large discrepancy between the initial and true values of λ. By dynamically adjusting the optimization process, this approach mitigates issues such as loss component imbalance, which were evident in the non-adaptive approach.

Figure 4.21 illustrates the estimated λ progression across training epochs. Unlike the non-adaptive method, the predicted λ in the adaptive approach converges rapidly

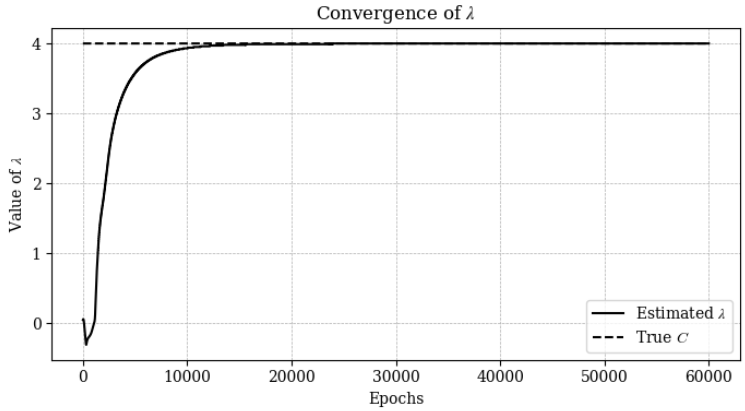


Figure 4.21: Adaptive technique - λ estimation - inverse problem - (λ far away from real valu)

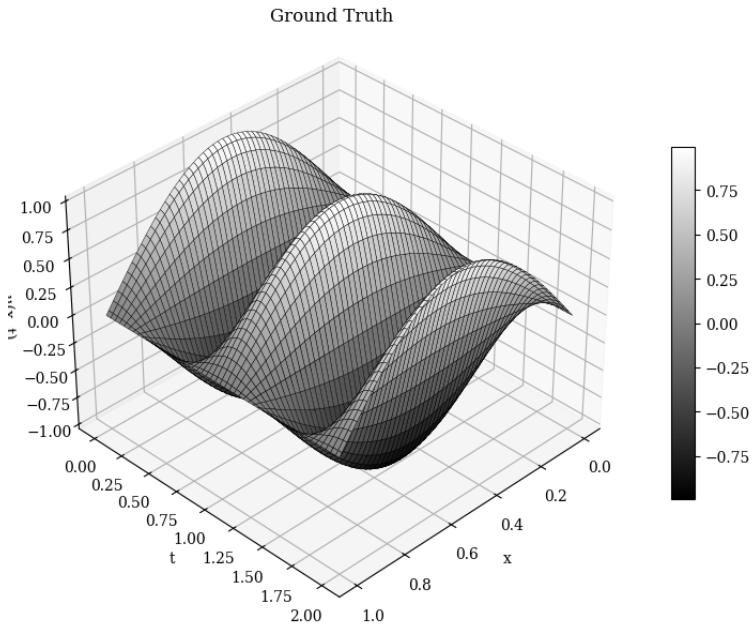


Figure 4.22: Adaptive technique - Ground truth of inverse problem -(λ far away from real value)

towards the true value of 4.0. The adaptive method ensures that the optimization landscape becomes smoother and more conducive to convergence by dynamically balancing the contributions of different loss components. This is evident in the near-perfect alignment between the true and predicted values of λ after a relatively small number of training epochs. The results underscore the robustness of the adaptive method in handling inverse problems with challenging initial parameter settings.

The comparison between the exact state (Figure 4.23) and the PINN-predicted state reveals a high degree of agreement, with the adaptive method successfully capturing the intricate dynamics of the solution. Unlike the non-adaptive approach, where discrepancies were apparent in regions of high gradients, the adaptive method effectively

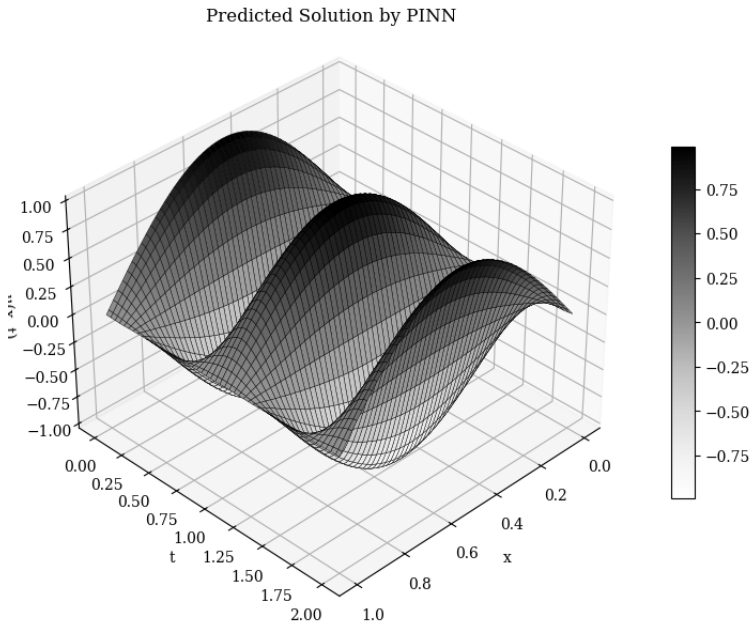


Figure 4.23: Adaptive technique -Prediction of inverse problem -(λ far away from real value)

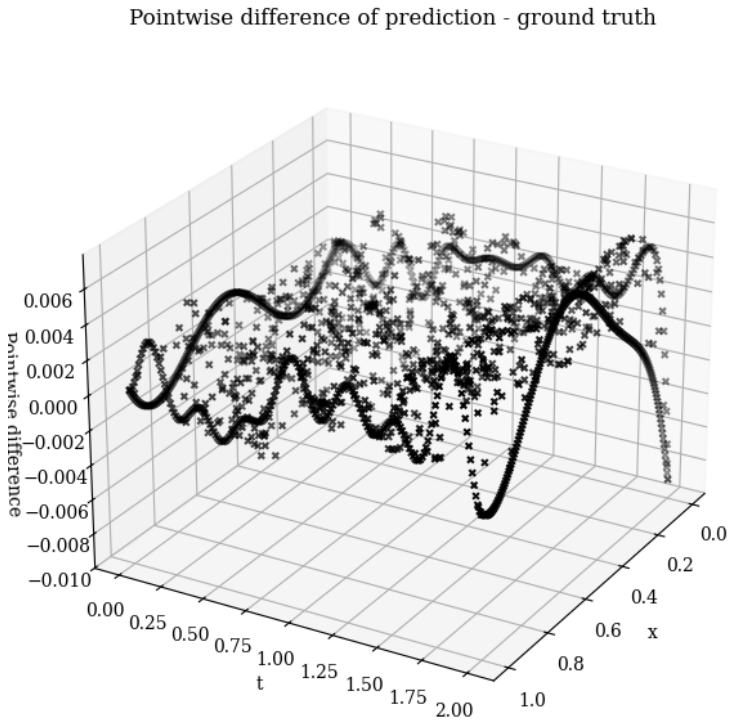


Figure 4.24: Adaptive technique - Pointwise difference between PINN prediction and ground truth - (λ far away from real value)

resolves these regions, producing a solution that closely resembles the ground truth. This improvement highlights the critical role of adaptive optimization in enabling PINNs to learn complex solutions governed by PDEs.

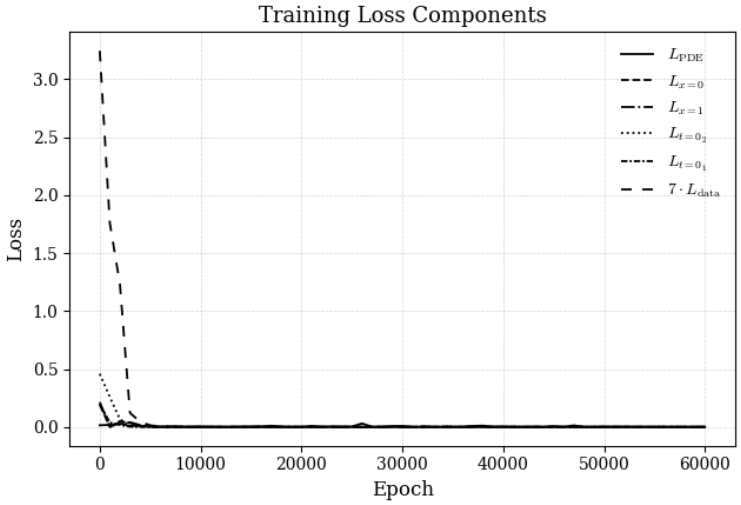


Figure 4.25: Adaptive technique - Components of loss functions - inverse problem -(λ far away from real value)

Figure 4.24 shows the pointwise difference between the PINN-predicted state and the exact state. The error is minimal and uniformly distributed across the domain, demonstrating the success of the adaptive method in minimizing localized errors. This uniformity is particularly important in applications where solution accuracy must be maintained across all regions of the domain. The adaptive approach achieves this by dynamically prioritizing areas of the domain where the model exhibits higher errors during training, thereby ensuring more effective learning.

Figure 4.25 provides a breakdown of the loss components over the training epochs. The adaptive method ensures that all loss components converge smoothly, with no single component dominating the optimization process. This balance is a critical advantage of the adaptive approach, as it prevents overfitting to specific constraints (e.g., initial or boundary conditions) at the expense of the governing PDE. The rapid convergence of the PDE residual loss, L_{PDE}, indicates that the adaptive weighting strategy successfully enforces the physics constraints, enabling the model to recover λ and the state solution accurately.

Overall, the adaptive method addresses the key limitations of the non-adaptive approach by dynamically adjusting the optimization process to suit the demands of the problem. The significant improvements in λ recovery, solution accuracy, and loss convergence underscore the effectiveness of this approach. Future extensions could explore more sophisticated adaptive strategies, such as meta-learning or reinforcement learning, to further enhance the robustness and generalizability of PINNs for solving challenging inverse problems.

Chapter 5

Conclusion

The comparative analysis of the forward problem, inverse problem, and adaptive inverse problem conducted in this thesis underscores the versatility and challenges of using PINNs for solving PDEs. For the forward problem, the PINN framework effectively approximated the solution of the 1D wave equation with Dirichlet boundary conditions. The network achieved a remarkable average Mean Squared Error (MSE) of 0.001, demonstrating its capability to capture the underlying dynamics of the system accurately. The configuration of the network, with four hidden layers comprising 50 nodes each, the `tanh` activation function, and the L-BFGS-B optimizer, enabled smooth convergence during training and facilitated the accurate prediction of the solution. The comparison between the PINN state and the exact state confirmed that the forward PINN model successfully resolved the wave-like physical behavior of the system with minimal error.

Conversely, the inverse problem revealed the inherent sensitivity of PINNs when tasked with parameter estimation, particularly for the rod parameter λ. While the standard PINN approach succeeded in estimating λ for favorable initial conditions, its performance diminished significantly when the initial guess was far from the true value. The inability to converge under these conditions highlighted the challenges posed by poorly conditioned optimization landscapes and an imbalance in the contributions of various loss components. The standard PINN model produced localized errors, particularly in regions of high gradients or oscillatory behavior, where inaccuracies in λ estimation amplified throughout the solution. This limitation emphasized the need for more sophisticated optimization strategies to improve the reliability and robustness of PINNs for inverse problems.

The adaptive approach introduced in this thesis effectively addressed these limitations. By dynamically adjusting the weights of the loss function components during training, the adaptive PINN achieved a more balanced optimization process. The adaptive method rapidly converged to the true value of λ, even when the initial guess

deviated significantly. This method minimized localized errors, ensured uniform accuracy across the domain, and significantly reduced the prediction error for inverse problems. The adaptive approach demonstrated superior performance in capturing the intricate dynamics of the system and enforcing physics constraints, as evidenced by the rapid convergence of the PDE residual loss and the improved accuracy of λ estimation.

In conclusion, the experiments in this chapter illustrate the strengths and limitations of PINNs for solving forward and inverse problems governed by PDEs. While the forward problem highlighted the robustness of PINNs in directly approximating solutions, the inverse problem demonstrated the sensitivity of parameter estimation. The adaptive PINN method bridged this gap, providing an effective framework for enhancing model performance in inverse problem settings.

Chapter 6

Future Work

In the realm of PINNs, several avenues for future research present themselves:

1. **Enhancing Computational Efficiency**: Addressing the high computational costs associated with training PINNs is crucial. Developing more efficient algorithms and leveraging advanced hardware can facilitate the application of PINNs to larger, more complex systems ([13]).
2. **Improving Accuracy and Convergence**: Investigating methods to enhance the precision and convergence rates of PINNs is essential. This includes exploring novel optimization strategies and network architectures ([33]).
3. **Expanding Application Domains**: Extending the use of PINNs to new fields, such as power systems and material science, can uncover novel applications and challenges ([37]).
4. **Integrating with Traditional Numerical Methods**: Combining PINNs with established numerical techniques may yield hybrid models that capitalize on the strengths of both approaches, potentially leading to more robust solutions ([10]).
5. **Addressing Data Scarcity**: Developing strategies to effectively train PINNs with limited data is vital, especially in scientific and engineering domains where data collection is challenging.
6. **Exploring Causality in Models**: Investigating how to embed causal relationships within PINNs can lead to models that better reflect underlying physical processes ([13]).

By pursuing these research directions, the potential of PINNs can be further realized, leading to more accurate and efficient solutions across various scientific and engineering disciplines.

Bibliography

[1] Amirhossein Arzani and Scott T. M. Dawson. “Data-driven cardiovascular flow modelling: examples and opportunities”. In: *Journal of The Royal Society Interface* 18.175 (2021). doi: 10.1098/rsif.2020.0802. url: https://doi.org/10.1098%2Frsif.2020.0802.

[2] Amirhossein Arzani, Jian-Xun Wang, and Roshan M. D'Souza. “Uncovering near-wall blood flow from sparse data with physics-informed neural networks”. In: *Physics of Fluids* 33.7 (2021), p. 071905. doi: 10.1063/5.0055600. url: https://doi.org/10.1063%2F5.0055600.

[3] G. K. Batchelor. “An introduction to fluid dynamics (2nd pbk. ed.)” In: *Cambridge University Press* ().

[4] Atilim Gunes Baydin et al. “Automatic differentiation in machine learning: a survey”. In: (2015). doi: 10.48550/ARXIV.1502.05767. url: https://arxiv.org/abs/1502.05767.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[6] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] Tim De Ryck, Ameya D. Jagtap, and Siddhartha Mishra. “Error estimates for physics informed neural networks approximating the Navier-Stokes equations”. In: (2022). doi: 10.48550/ARXIV.2203.09346. url: https://arxiv.org/abs/2203.09346.

[8] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009), pp. 248–255.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Ehsan Haghighat et al. “A Review of Physics Informed Neural Networks for Multiscale Analysis”. In: *Computational Mechanics* 69 (2024), pp. 1–16. doi: 10.1007/s42493-024-00106-w. url: https://link.springer.com/article/10.1007/s42493-024-00106-w.

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. issn: 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. url: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[12] iDocPub contributor. *Longitudinal Vibration Bars*. Accessed: 2025-07-01. n.d. url: https://idoc.pub/documents/longitudinal-vibration-bars-6nge0q7zm1lv.

[13] Ehsan Kharazmi, Zongyi Zhang, and George Em Karniadakis. "Physics-Informed Neural Network (PINN) Evolution and Beyond: A Survey". In: *Big Data and Cognitive Computing* 6.4 (2022), p. 140. doi: 10.3390/bdcc6040140. url: https://www.mdpi.com/2504-2289/6/4/140.

[14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2014.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[16] I.E. Lagaris, A. Likas, and D.I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. doi: 10.1109/72.712178. url: https://doi.org/10.1109%2F72.712178.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[18] Hyuk Lee and In Seok Kang. "Neural algorithm for solving differential equations". In: *Journal of Computational Physics* 91.1 (1990), pp. 110–131. issn: 0021-9991. doi: https://doi.org/10.1016/0021-9991(90)90007-N. url: https://www.sciencedirect.com/science/article/pii/002199919090007N.

[19] Dong C Liu and Jorge Nocedal. "Limited memory BFGS for bound constrained optimization". In: *Mathematical Programming* 45.1-3 (1989), pp. 503–528.

[20] Stefano Markidis. "The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?" In: (2021). doi: 10.48550/ARXIV.2103.09655. url: https://arxiv.org/abs/2103.09655.

[21] A.J. Meade and A.A. Fernandez. "The numerical solution of linear ordinary differential equations by feedforward neural networks". In: *Mathematical and Computer Modelling* 19.12 (1994), pp. 1–25. issn: 0895-7177. doi: https:

//doi.org/10.1016/0895-7177(94)90095-7. url: https://www.sciencedirect.com/science/article/pii/0895717794900957.

[22] Siddhartha Mishra and Roberto Molinaro. "Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating a class of inverse problems for PDEs". In: (2020). doi: 10.48550/ARXIV.2007.01138. url: https://arxiv.org/abs/2007.01138.

[23] Aliki D Mouratidou and Georgios E Stavroulakis. "A Gentle Introduction to Physics-Informed Neural Networks, with Applications in Static Rod and Beam Problems". In: *Journal of Advances in Applied Computational Mathematics* (2022).

[24] Aliki D Mouratidou and Georgios E Stavroulakis. "Ensemble of Physics-Informed Neural Networks for Solving Plane Elasticity Problems with Examples". In: *International Journal of Computational Methods and Experimental Measurements* (2023).

[25] Aliki D Mouratidou and Georgios E Stavroulakis. "Nonlinear Interaction in Composites Using Physics Informed Neural Networks". In: *16th World Congress on Computational Mechanics*. 2024.

[26] Sinno Jialin Pan and Qiang Yang. "A survey on transfer learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2009), pp. 1345–1359.

[27] Wei Peng et al. "IDRLnet: A Physics-Informed Neural Network Library". In: (2021). doi: 10.48550/ARXIV.2107.04320. url: https://arxiv.org/abs/2107.04320.

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

[29] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations". In: (2017). doi: 10.48550/ARXIV.1711.10561. url: https://arxiv.org/abs/1711.10561.

[30] Maziar Raissi, Alireza Yazdani, and George Karniadakis. "Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data". In: (Aug. 2018).

[31] Maziar Raissi, Alireza Yazdani, and George Karniadakis. “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations”. In: *Science* 367 (Jan. 2020), eaaw4741. doi: 10.1126/science.aaw4741.

[32] Dania Sana. *FAUMoD InternReport: PINN*. Internal Report. Accessed: 2025-07-01. Department of Computer Science, Friedrich–Alexander–Universität Erlangen–Nürnberg (FAU), 2021. url: https://dcn.nat.fau.eu/wp-content/uploads/FAUMoD_DaniaSana-InternReport_PINN.pdf.

[33] Edward Small. “An Analysis of Physics-Informed Neural Networks”. In: (2023). doi: 10.48550/arXiv.2303.02890. url: https://arxiv.org/abs/2303.02890.

[34] G.E. Stavroulakis et al. “A neural network approach to the modelling, calculation and identification of semi-rigid connections in steel structures”. In: *Journal of Constructional Steel Research* 44.1 (1997). Structural Steel Research in Greece, pp. 91–105. issn: 0143-974X. doi: https://doi.org/10.1016/S0143-974X(97)00039-4. url: https://www.sciencedirect.com/science/article/pii/S0143974X97000394.

[35] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural Networks for Machine Learning* (2012).

[36] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[37] Junbo Zhang et al. “Applications of Physics-Informed Neural Networks in Power Systems: A Review”. In: *IEEE Transactions on Power Systems* 37.4 (2022), pp. 3271–3285. doi: 10.1109/TPWRS.2022.3159123. url: https://ieeexplore.ieee.org/document/9743327.