TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Maschinenwesen, Institut für Luft- und Raumfahrttechnik, Professur für Luftfahrzeugtechnik

Master Thesis

ILR-LFT MA 23-01

Physics Informed Neural Networks for Complex Geometries

Ruchit Naresh Kini

To
obtain the academic degree

Master of Science
(M.Sc.)

Supervisor:	M.Sc. Javed Arshad Butt
Responsible university lecturer:	Prof. Dr. Johannes Markmiller
Date of submission:	14.05.2024
First reviewer:	Prof. Dr. Johannes Markmiller
Second reviewer	Dr.-Ing. Falk Hähnel

Declaration of Independence

Hereby, I declare that I am submitting to the chair of Aircraft Engineering my master thesis titled

Physics Informed Neural Networks for complex geometries

I wrote the report independently and did not use any resources other than those specified in citations.

Dresden, 14. May 2024

Ruchit Naresh Kini

Abstract

Physics Informed Neural Networks for complex geometries

Partial differential equations (PDEs) are vital in mathematical physics and engineering, but their complex nature often requires numerical methods for practical solutions. Machine learning techniques, particularly Physics Informed Neural Networks (PINNs), have emerged as effective tools for solving challenging PDEs, integrating physical principles into neural network architectures. This thesis explores PINNs for solving linear elasticity equations in two-dimensional and three-dimensional problems using the DeepXDE library. The study assesses the capability of PINNs to handle complex boundary conditions and geometries, comparing their performance with Finite Element Method (FEM) simulations. Key aspects evaluated include convergence, accuracy, and computational efficiency. The thesis aims to evaluate the efficacy of PINNs in addressing complex problem-solving challenges, presenting them as a promising alternative to traditional numerical methods. The research focuses on optimizing PINN training through network design and hyperparameter tuning to improve their utility and scalability in engineering and physics applications. Future work aims to broaden PINN capabilities for practical modeling tasks in real-world scenarios.

Kurzfassung

Physikinformierte neuronale Netze für komplexe Geometrien

Partielle Differentialgleichungen (PDEs) sind in der mathematischen Physik und im Ingenieurwesen von entscheidender Bedeutung, aber ihre komplexe Natur erfordert oft numerische Methoden für praktische Lösungen. Maschinelle Lerntechniken, insbesondere Physics Informed Neural Networks (PINNs), haben sich als effektive Werkzeuge zur Lösung anspruchsvoller PDEs erwiesen, indem sie physikalische Prinzipien in neuronale Netzwerkarchitekturen integrieren. In dieser Arbeit werden PINNs zur Lösung von linearen Elastizitätsgleichungen in zwei- und dreidimensionalen Problemen unter Verwendung der DeepXDE-Bibliothek untersucht. Die Studie bewertet die Fähigkeit von PINNs, komplexe Randbedingungen und Geometrien zu bewältigen, und vergleicht ihre Leistung mit Simulationen nach der Finite-Elemente-Methode (FEM). Zu den bewerteten Schlüsselaspekten gehören Konvergenz, Genauigkeit und Berechnungseffizienz. Ziel der Arbeit ist es, die Wirksamkeit von PINNs bei der Lösung komplexer Probleme zu bewerten und sie als vielversprechende Alternative zu traditionellen numerischen Methoden zu präsentieren. Die Forschung konzentriert sich auf die Optimierung des PINN-Trainings durch das Netzwerkdesign und die Abstimmung der Hyperparameter, um ihren Nutzen und ihre Skalierbarkeit in technischen und physikalischen Anwendungen zu verbessern. Zukünftige Arbeiten zielen darauf ab, die PINN-Fähigkeiten für praktische Modellierungsaufgaben in realen Szenarien zu erweitern.

Contents

List of Symbols

Latin Symbols	Unit	Meaning
E	Pa	Young's Modulus
p	Pa	Pressure
$\mathbf{g}$	m/s^2	Gravitational acceleration
t	s	Time
$\bar{t}$	N/m^2	Traction vector
n	-	Unit normal vector
C	N/m^2	Cauchy stress tensor

Greek Symbols	Unit	Meaning
ν	-	Poisson's ratio
λ	N/m^2	Lame's first parameter
μ	N/m^2	Lame's second parameter
σ	N/m^2	Stress
ϵ	-	Strain
ρ	kg/m^3	Density
δ_{ij}	-	Kronecker's delta

Indices	Unit	Meaning
L, l	m	Length
H	m	Height

Miscellaneous Symbols	Meaning
$\otimes$	Tensor product
$\sum$	Summation
Δ	Laplacian Operator
∇	Nabla Operator

Abbreviations	Meaning
PDE	Partial Differential Equation
PINN	Physics Informed Neural Network
ANN	Artificial Neural Network
FNN	Feed forward Neural Network
CNN	Convolutional Neural Network
GNN	Graph Neural Network
MLP	Multi-Layer Perceptron
MFNN	Multi-Feedforward Neural Network
SGD	Stochastic Gradient Descent
MSE	Mean Squared Error
CFD	Computational Fluid Dynamics
FEM	Finite Element Method
VLM	Vortex Lattice Method
SPINN	Seperable Physics Informed Neural Network

1 Introduction

Partial differential equations (PDEs) have stood as foundational elements in mathematical physics and engineering design for centuries, dating back to the inception of the one-dimensional wave equation by d'Alembert in 1752 [19] [20]. PDEs offer a structured mathematical framework for describing the variations of quantities across multiple variables, commonly space and time. They are a fundamental tool for articulating the governing equations of various spatio-temporal physical phenomena such as solid mechanics, electrodynamics, quantum mechanics, fluid mechanics, and heat transfer.

Obtaining analytical solutions for PDEs is challenging due to nonlinearities, complex boundary conditions, and irregular geometries, making closed-form solutions [53] impractical in many cases. As a result, numerical or approximate methods are commonly used to model and analyze complex physical systems. These methods include finite difference, finite element, and spectral methods, which discretize the problem domain to approximate solutions at discrete points or regions. While effective, traditional methods face challenges with non-linearities and intricate geometries, and their reliance on explicit mathematical formulations limits their applicability. Dealing with complex models often leads to significant computational challenges and longer simulation times. Addressing this challenge often involves optimizing numerical methods, utilizing high-performance computing resources, and exploring alternative simulation techniques, such as reduced-order modeling [41] or machine-learning approaches.

1.1 Machine Learning Approach

The emergence of Machine Learning (ML) has reshaped computational modeling by offering data-driven alternatives to standard methods. ML techniques, particularly neural networks, shine at capturing complex patterns and relationships from data, making them ideal for addressing nonlinear and high-dimensional problems. ML has augmented traditional methods by enabling the event of surrogate or reduced-order models [41] that accurately predict complex system behaviors with significantly reduced computational costs compared to traditional simulations. This integration has transformed computational modeling, blending traditional approaches' robustness with ML algorithms' adaptability and scalability. As a result, researchers and engineers can efficiently tackle diverse challenges

across fields like engineering, physics, biology, and beyond with unprecedented efficiency and accuracy.

Machine Learning, particularly through artificial neural networks, has transformed computational modeling by efficiently approximating complex, nonlinear functions and capturing underlying patterns in data for tasks like regression and classification. Neural networks have proven highly effective in approximating a broad range of PDEs with remarkable accuracy and efficiency. Physics-Informed Neural Networks (PINNs) exemplify this application by integrating physics principles into neural network architectures. PINNs excel in solving PDEs and simulating physical phenomena with exceptional accuracy and efficiency by embedding domain-specific knowledge, like conservation laws and boundary conditions. PINNs are especially adept at handling complex geometries and boundary conditions without explicit meshing, making them ideal for problems involving irregular boundaries, free surfaces, or multiphase flows.

While PINNs offer several benefits, they also encounter challenges of suitable specification and hyperparameter tuning, high computational costs, extrapolation limitations, and reduced interpretability compared to traditional methods. Ongoing research aims to mitigate these issues through techniques like regularization and hardware advancements. Despite challenges, PINNs represent a promising approach for solving complex physical problems, integrating physics principles directly into the training process for enhanced accuracy and efficiency in fields spanning fluid dynamics to materials science. By understanding the underlying concepts and methodologies of PINNs, researchers can unlock new possibilities for innovation and discovery within the realm of physical sciences and engineering.

1.2 Problem Statement and Objective

As outlined in the previous section, the limitations of conventional methods and the advantages offered by modern ML techniques highlight the need to explore the use of Neural Networks, particularly Physics-Informed Neural Networks, for modeling and training. This motivation underpins the chosen thesis topic, which seeks to enhance the effectiveness of approximating solutions to PDEs.

This thesis aims to develop and validate the PINN model capable of training complex geometries based on specified governing equations. The objective is to validate the model

outcomes by comparing them with results from FEM software or analytical solutions, where applicable. The PINN model is implemented using Python with the DeepXDE library and PyTorch as the backend framework. Initially, the focus is on building basic models, aiming to understand the implementation of PINN methodology, neural network parameters, and optimization techniques. The scope then narrows to PDEs related to linear elasticity, emphasizing the training of simple two-dimensional geometries on linear elasticity equations. Various problem cases involving different geometrical shapes and boundary conditions are considered to gain a comprehensive understanding of PINNs. The predicted results from the PINN model are subsequently compared with reference solutions obtained from FEM software.

Following this, the methodology is extended to encompass three-dimensional geometries, updating the linear elasticity equations accordingly. Similar evaluations and comparisons between the PINN model and the FEM model are conducted. The next objective is to demonstrate the PINN model's capability to predict solutions for realistic problems, such as deformation and stress analysis on aircraft wings subjected to lift forces. While the aircraft wing model is simplified to a single aircraft panel for the thesis scope, the geometry retains the three-dimensional characteristics of an airfoil. Correspondingly, similar dimensions are utilized in the CFD tool to obtain lift forces, which are subsequently applied to both the PINN model and FEM Model.

Lastly, the thesis aims to address the challenges encountered during the training of the PINN model, highlight its limitations, and propose potential future steps to mitigate these issues.

1.3 Structural Overview: Chapters and Sections Breakdown

The thesis is organized as follows:

- Chapter 1, provides an overall perspective on the research subject, outlining its motivation and defining the thesis problem.
- Chapter 2, delves into fundamental concepts. The chapter is segmented into sections covering artificial neural networks, neural network architecture, the principles of physics-informed neural networks, and the governing equations relevant to the problem domain.

- Chapter 3, details the methodologies and software libraries utilized in this study. It encompasses discussions on the neural network architecture and its implementation within the DeepXDE library. Additionally, it outlines the process of mapping forces from the CFD tool to both the PINN model and the FEM model.
- Chapter 4, the assessment of results across various geometries is conducted, addressing their accuracy, convergence challenges, and the efficacy of the PINN model.
- Chapter 5, the final chapter presents a comprehensive summary of the undertaken work and offers insights into potential future research directions.

2 Background

This chapter begins with an overview of the fundamentals of artificial neural networks in the first section, which includes discussions on their types and architectures. Following that, the subsequent section introduces physics-informed neural networks, outlining their fundamental concepts and discussing various methods for optimizing the objective function. Finally, the chapter delves into the governing equations of linear elasticity, presenting them in various notations and exploring their application in two and three-dimensional scenarios, accompanied by a detailed discussion on different boundary conditions.

2.1 Artificial Neural Network (ANN)

An Artificial Neural Network (ANN) is a computational framework inspired by the structure of biological neural network [28]. While an ANN draws inspiration from biological neural networks, they are significantly simplified and do not replicate the complete complexity observed in biological neural networks. By definition, an ANN is "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs" - Dr. Robert Hecht-Nielson as quoted in “Neural Network Primer: Part I” by Maureen Caudill, AI Expert, Feb. 1989. A neuron is a fundamental unit of a neural network, mimicking the basic structure and function of a biological neuron that processes and transmits information through electrochemical signals. A neural network constitutes a system of interconnected neurons arranged in layers. It uses these neurons to process input data, perform complex computations, and generate output. While a neuron is an individual unit, a neural network is a collection of interconnected neurons working together to perform tasks such as pattern recognition, classification, and regression. A biological neuron and an equivalent artificial neuron (often called a node) are illustrated in figure 2.1 and figure 2.2.

A single perceptron is the fundamental building block of artificial neural networks, serving as the simplest form of a neural network unit and is often referred to as a neuron or a node. The perceptron receives inputs, typically represented as input values $x_1, x_2, \ldots, x_n$. Each input value is associated with a weight, $w_1, w_2, \ldots, w_n$, which signifies its importance or contribution to the output. The perceptron computes the weighted sum of the input signals and their corresponding weights. This operation represents the linear combination

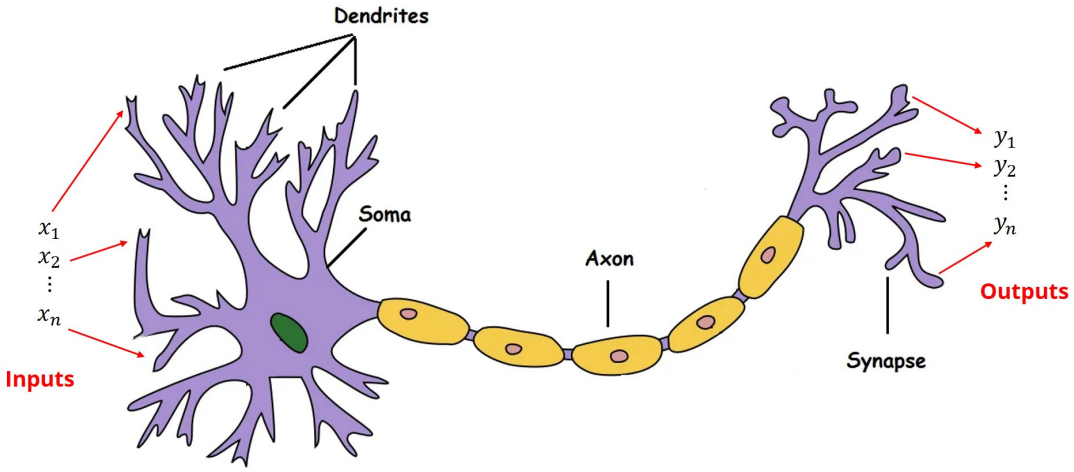


Figure 2.1: A simplified representation of a biological neuron (*adapted from* [4] [11])

of inputs and weights and is calculated as [15],

$$u = w_1 \times x_1 + w_2 \times x_2 + ... + w_n \times x_n \tag{2.1}$$

The weighted sum is passed through an activation function. The activation function introduces non-linearity to the model and determines whether the perceptron should fire or remain inactive based on the computed value of u. The output of the activation function becomes the output of the perceptron. During the learning phase, the perceptron adjusts its weights based on the error in its predictions. The process involves comparing the predicted output with the actual output (target) and updating the weights accordingly using techniques like the perceptron learning algorithm or gradient descent. In addition to input signals and weights, a perceptron often includes a bias term, which represents the threshold for activation. The bias allows the perceptron to make predictions even when all input signals are zero. The complete calculation for a single perceptron is expressed as follows [15],

$$\bar{u}_N = f_a \left(\sum_i^n w_i x_i + b \right) \tag{2.2}$$

where, $\bar{u}_N$ is the output of the neuron, f_a is the activation function, x_i is the input, w_i is the weight associated with the input connection, n is the number inputs and b is the bias value.

An illustration of a single perceptron and the mathematical representation of calculation is shown in figure 2.2.

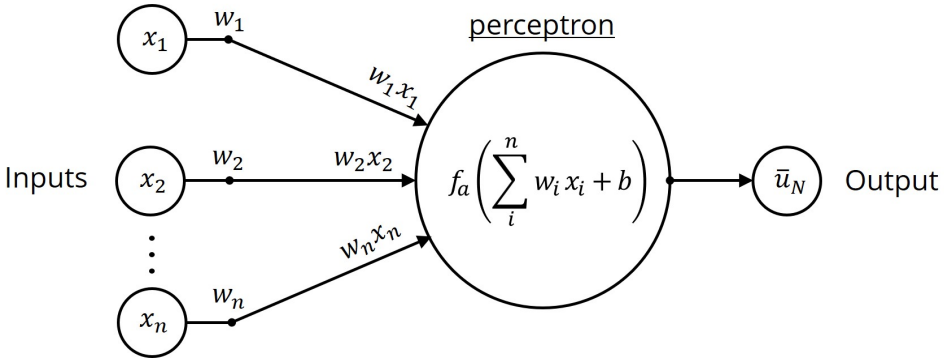


Figure 2.2: Computation on single perceptron.

2.1.1 Neural Network Architecture

Neural networks encompass a diverse range of architectures and training strategies, each tailored to specific problem domains and model complexities [13] [42] [44] [52] [61]. These architectures often feature distinct network structures and training methodologies. In this section, several neural network architectures are covered, including Feed Forward Neural Networks (FFNN), Convolutional Neural Networks (CNN), and Graph Neural Networks (GNN), among the diverse range available.

A fully connected feedforward network is known for its simplicity and versatility in addressing various problem types. A feed-forward network comprises interconnected layers of neurons, where information flows in one direction, from the input layer through hidden layers to the output layer. At each layer, computations are performed on a node based on weighted inputs, bias, and activation functions. It can range from a single-layer perceptron, consisting of a single-layer computation node (figure 2.2), to a multilayer perceptron (MLP) with multiple layers and nodes, as illustrated in figure 2.3. In an MLP, each neuron in a layer is connected to every neuron in the subsequent layer, enabling the network to learn complex relationships between inputs and outputs through the hierarchical arrangement of neurons.

Convolutional Neural Networks, or CNNs are primarily used for processing multidimensional array-like data, such as images, speech signals, and spatially structured inputs [22]. CNNs leverage several convolution and pooling layers to extract hierarchical features from input data, enabling them to capture spatial patterns and hierarchies effectively. They have found widespread applications in image classification [50], object detection [29], and image segmentation tasks [54].

Graph Neural Networks, or GNNs, are designed for handling graph-structured data, such as social networks, molecular structures, and recommendation systems. Graphs are mathematical representations of nodes connected by edges, GNNs operate directly on graph representations, allowing them to learn and reason about relationships between entities in complex networks [36]. GNNs have the inherent ability to learn and reason about graph-structured data, enabling diverse applications involving graph classification [45], node classification [57], and link prediction [64].

While fully connected feed-forward networks serve as a fundamental building block of neural network architectures, specialized models like CNNs and GNNs offer tailored solutions for processing specific types of data and addressing domain-specific challenges. The choice of neural network architecture depends on the nature of the data, the complexity of the problem, and the desired outcomes of the task at hand. For this thesis, MLPs have been used and in the next section, we will discuss the architecture of an MLP in detail.

2.1.2 Multilayer Perceptron (MLP)

A Multilayer Perceptron (MLP) is a type of fully connected feed-forward network. The neurons in MLP are connected layer to layer but not within the layer. The flow of data is only in the forward direction without any loops. An MLP comprises interconnected layers of nodes, with each connection having an associated weight. During training, these weights and bias values are adjusted iteratively based on the model's output accuracy. The architecture of an MLP is illustrated in figure 2.3. The essential characteristics of neural network architecture and the working of an MLP are as follows:

- Input nodes and input layer: A block of nodes is termed as a layer. The input layer consists of all the input nodes ($x_1, x_2, \ldots x_n$). No computation takes place in this layer as the node values are just inputs that are passed to the hidden layers or output layer.
- Hidden nodes and hidden layers: The hidden layer is the intermediate layer situated between the input layer and the output layer and conducts calculations on the nodes. A neural network can have no hidden layer, one hidden layer, or more than one hidden layer depending on the complexity of the problem. The values computed by the nodes are subsequently transmitted to the next layer, which can be either another hidden layer or the output layer.

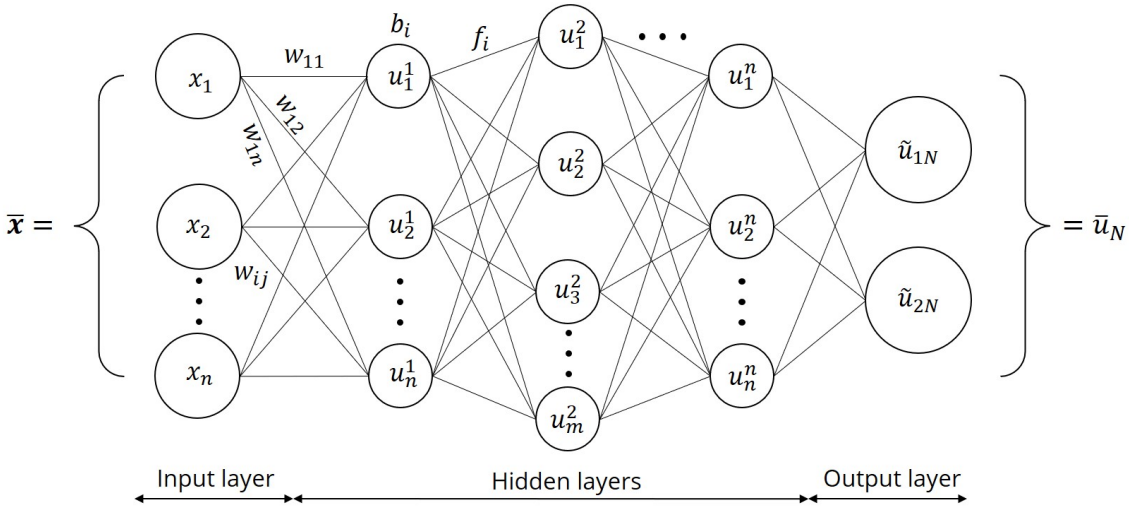


Figure 2.3: Neural Network Architecture of an MLP

- Output nodes and output layer: The output layer is the final layer consisting of output nodes that receive the weighted inputs. The output layer does not have an activation function so no transformation takes place. The output layer can be further scaled to the desired output range.
- Connections, weights, and bias: Each node has a connection with other nodes for transferring the information. These connections transfer the output from the previous node i to successive node j as an input. Every connection that connects node i to node j is assigned a weight, w_{ij}.
- Activation function: An activation function is a mathematical function applied to the output of each neuron in a neural network layer introducing a non-linearity to the network, allowing it to learn complex patterns in the data. The activation function determines whether a neuron should be activated or not, based on whether the neuron's input is relevant to the model's prediction. There are several common activation functions used in neural networks like "sigmoid", "tanh", "ReLU", and "Leaky ReLU" [2]. The hyperbolic tangent (tanh) is a commonly used activation function known for its nonlinearity and continuous differentiability and is given as,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

These are just a few examples, and there are other activation functions as well [24]. The choice of activation function depends on the specific task and characteristics of

the data. For our case, we have used "tanh" as the activation function.

- Learning rule: The learning rule [9] refers to an algorithm or mechanism that adjusts the parameters of a neural network, such as weights and thresholds, to ensure that when a specific input is provided to the network, it generates the desired output. This adjustment process involves modifying the network's parameters to optimize its performance in producing the desired outputs.

In an MLP, the computation on each node can be represented mathematically as [22],

$$f_i(x_i; W_i, b_i) = \alpha_i(W_i \cdot x_i + b_i) \tag{2.4}$$

where x_i is input to layer i or output of $(i-1)^{th}$ layer, W_i and b_i are weights and biases of the i^{th} layer respectively, α_i is the non-linear activation function. For identical activation functions in all the layers, MLP with K layers can be represented in a generic notation as [22],

$$u_\theta(x) = C_K \circ \alpha \circ C_{K-1} \circ \cdots \circ \alpha \circ C_1(x), \tag{2.5}$$

where $C_k(x_k)$ is defined as [22],

$$C_k(x_k) = W_k x_k + b_k, \qquad \text{for any } 1 \leq k \leq K \tag{2.6}$$

The neural network has an input and output layer and $(K-2)$ hidden layers.

2.2 Physics Informed Neural Networks (PINNs)

Physics-informed Neural Networks (PINNs) is a scientific machine-learning technique that is used to solve problems involving Partial Differential Equations (PDEs). PINNs consider the physics involved in the problems and approximate the solution of the PDEs by training the neural network to minimize the loss function [22]. The loss function consists of loss terms for initial conditions, boundary conditions along the space-time domain boundary, and PDE residual for the selected points in the domain (also called collocation points).

The PINN approach is different from the standard ML approach. It can be thought of as an unsupervised ML strategy as it does not require any labeled data from previous observations and purely depends on physical laws and constraints to guide the training process.

A comprehensive depiction of PINNs is provided in figure 2.4. A PINN comprises a neural network responsible for approximating the output based on weight initialization. This output is subsequently fed into another network, which enforces the physical constraints. These constraints typically represent the governing equations of the physical problem along with the associated initial and boundary conditions. The approximated output from the neural network is used to evaluate the residual values of PDEs, as well as the residuals arising from initial and boundary conditions. Automatic differentiation [18] is employed to compute the derivatives of the approximated values, facilitating the calculation of residuals. The total loss is then computed by summing the losses arising from the PDE residuals, initial condition residuals, and boundary condition residuals. The objective is to minimize this total loss until the desired level of accuracy is attained. The figure elucidates the general approach of PINNs. Further elaboration on the mathematical formulation of PINNs for PDEs will be provided in subsequent sections.

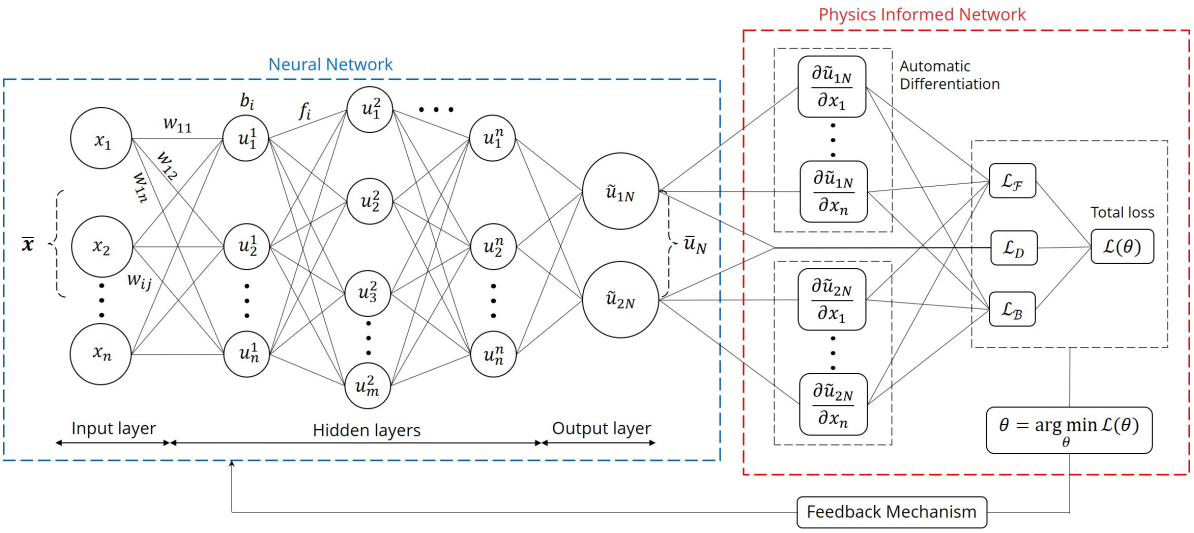


Figure 2.4: Schematic representation of Physics Informed Neural Network Method

2.2.1 Building Blocks

PINNs are versatile tools capable of solving a wide range of PDEs that arise in various scientific and engineering domains. These equations include Diffusion equations (Equation 2.7) which describe the density fluctuations in a material undergoing diffusion [7]; wave equations (Equation 2.8) which are second-order PDEs for the description of waves or standing wave fields [31]; Navier-Stokes equations (Equation 2.9) which describe the motion of vis-

cous fluid substances and are widely used in the field of fluid dynamics [27]; and equations governing the linear elasticity problems, the primary focus of this thesis. These are just a handful of examples; PINNs can solve a wide range of additional PDEs. The diffusion equation is given as [7],

$$\frac{\partial \phi(\mathbf{r}, t)}{\partial t} = \nabla \cdot \big[D(\phi, \mathbf{r})\, \nabla \phi(\mathbf{r}, t)\big], \tag{2.7}$$

where $\phi(\mathbf{r}, t)$ is the density of the diffusing material at location $\mathbf{r}$ and time t and $D(\phi, \mathbf{r})$ is the collective diffusion coefficient for density ϕ at location $\mathbf{r}$.

The wave equation is given as [31],

$$\frac{\partial^2 u}{\partial t^2} = c^2 \Delta u, \tag{2.8}$$

where c is a fixed non-negative real coefficient, u represents displacement and Δ is the spatial Laplacian operator [8].

The Navier-Stokes equation is given as [27],

$$\frac{\partial}{\partial t}(\rho\, \mathbf{u}) + \nabla \cdot (\rho\, \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho\, \mathbf{f} \tag{2.9}$$

where ρ is the density, $\mathbf{u}$ is the flow velocity, p is the pressure, t is time, $\boldsymbol{\tau}$ is the deviatoric stress tensor and $\mathbf{f}$ represents body forces.

PINNs offer a versatile framework for solving such a diverse range of PDEs. Their ability to integrate machine learning with physical principles makes them valuable tools for modeling and simulating complex physical systems. We will now see the mathematical formulation of PINNs for partial differential equations. A partial differential equation can be represented in a general form as [22],

$$\mathcal{F}(u(z); \gamma) = f(z), \quad z \text{ in } \Omega \tag{2.10}$$

$$\mathcal{B}(u(z)) = g(z), \quad z \text{ in } \partial\Omega \tag{2.11}$$

defined on the domain $\Omega \subset \mathbb{R}^d$ with the boundary $\partial\Omega$, where z is the coordinate vector, $u(z)$ is the unknown solution.

The solution of PDE, $u(z)$ is computationally predicted by the neural network, parametrized by a set of parameters θ, giving rise to the approximation, $\hat{u}_\theta(z) = u(z)$ where $\hat{(.)}$ denotes the solution approximated by the neural network with θ being the set of neural network

parameters.

The PDE residual is given as [22],

$$\mathcal{R}_{\mathcal{F}}[\hat{u}_\theta](z) = \mathcal{F}(u(z);\gamma) - f(z), \tag{2.12}$$

The PINN methodology determines parameters θ of the neural network by minimizing the loss function,

$$\theta = \arg\min \mathcal{L}(\theta) \tag{2.13}$$

The loss function or total loss, $\mathcal{L}(\theta)$ is the summation of all the loss terms from PDE residual and initial/boundary conditions,

$$\mathcal{L}(\theta) = w_{\mathcal{F}}\mathcal{L}_{\mathcal{F}}(\theta) + w_{\mathcal{B}}\mathcal{L}_{\mathcal{B}}(\theta) + w_D\mathcal{L}_D(\theta) \tag{2.14}$$

where, $\mathcal{L}_{\mathcal{F}}(\theta)$ is the loss from PDE residual, $\mathcal{L}_{\mathcal{B}}(\theta)$ is the loss from boundary condition, $\mathcal{L}_D(\theta)$ is the loss from the observed data (experimental or simulated data) if known. The parameters $w_{\mathcal{F}}$, $w_{\mathcal{B}}$, and w_D are the weights associated with every loss term.

The PINN methodology can be utilized to solve the forward problems that involve predicting system behavior based on known inputs, like geometry and boundary conditions. For example, this could mean predicting stress distribution in a material under specific loads. In contrast, inverse problems focus on estimating unknown inputs or properties like Young's modulus or Poisson's ratio based on observed outputs. PINNs address both by learning relationships between inputs and outputs to predict behaviors or infer unknown parameters from observed data. In our study, our main focus is solely on solving forward problems associated with linear elasticity equations.

2.2.2 Optimization Methods

Optimization is the process of finding the optimal solution or set of solutions for a given problem. In the context of mathematics, the goal of the optimization algorithm is to either maximize or minimize the objective function that satisfies the specific constraints. The optimization methods can vary depending on the problem definition and may include methods such as gradient descent, linear programming, genetic algorithms, and simulated annealing, among the numerous available methods [10]. A general classification of optimization

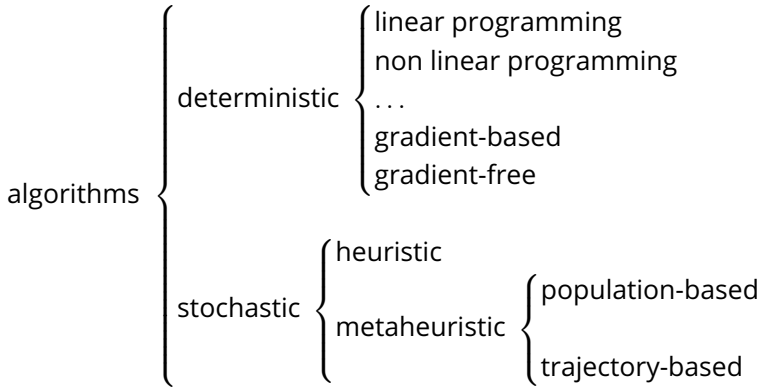


Figure 2.5: Classification of optimization algorithms [63]

methods [63] is illustrated in figure 2.5.

The deterministic algorithms utilize established mathematical equations to achieve the optimal solution and are typically employed in scenarios where the objective function and constraints are clearly defined. On the other hand, stochastic methods employ probabilistic algorithms to explore and search for the optimal solution, making them suitable for situations where the objective function is intricate, noisy, or challenging to evaluate directly. Among the deterministic categories, there are gradient-free [21] or direct optimization methods that iteratively explore the search space using heuristics or pattern-based strategies to find the optimal solution, and there are gradient-based techniques [49] [60], that utilize the gradients of the objective function, guiding the search for the optimal solution. Examples of gradient-free methods include the Nelder-Mead simplex method, genetic algorithms, and pattern search methods. In contrast, gradient-based algorithms encompass approaches like gradient descent, conjugate gradient descent, and Newton's methods. The choice between various available optimization methods depends on factors such as the problem's characteristics, computational resources, and the availability of gradient information.

In the context of neural networks, optimization refers to the process of adjusting the parameters of the network to minimize the error between the predicted output and the actual output for a given set of training data. The objective is to find the set of parameters that best fits the training data and generalizes well to unseen data. The optimizer minimizes the objective function often referred to as loss function, and updates the parameters of the neural network based on the gradient of the loss function concerning those parameters. Optimizers play a crucial role in training neural networks efficiently and effectively.

In neural networks, stochastic gradient-based optimization methods are predominantly used [56]. The primary reason for this preference is the availability of gradient information through backpropagation, which computes the derivatives of the loss function concerning the network parameters (weights and biases).

The optimization process in PINNs involves minimizing the loss function derived from the residuals of PDEs and the satisfaction of boundary and/or initial conditions. Among the available optimization methods [14], several commonly used ones in neural network training include gradient descent, stochastic gradient descent, ADAM, AdaGrad, RMSprop, and L-BFGS. In this section, only stochastic gradient descent, ADAM, and LBFGS will be discussed.

Stochastic Gradient Descent

The gradient descent method involves iteratively updating the model parameters based on the negative gradients of the loss function for those parameters over the full training set. The general form of a gradient descent algorithm is given as [49],

$$\theta := \theta - \alpha \nabla_\theta \mathbb{E}(J(\theta)) \tag{2.15}$$

where, θ is the parameter to be optimized such that it minimizes the loss function, $J(\theta)$; here ":=" denotes the update of a variable in the algorithm; α is the step size (learning rate) and $\mathbb{E}$ is the expectation of the loss function over the entire training set. The standard gradient descent algorithm performs computation on the entire data set for every iteration which makes it computationally expensive and slow. This drawback is overcome by the Stochastic Gradient Descend (SGD) [58] [48] algorithm. Each iteration of SGD computes the gradient based on one randomly chosen partition of the dataset which was shuffled, instead of using the whole part of the observations. This modification of gradient descent can reduce the computational time significantly. Mathematically,

$$\theta := \theta - \alpha \nabla_\theta J(\theta; x^{(i)}, y^{(i)}) \tag{2.16}$$

where, $x^{(i)}$ and $y^{(i)}$ are randomly chosen pair from the training set.

By processing data in mini-batches rather than the entire dataset, SGD offers computational efficiency. However, due to its stochastic nature, SGD may struggle to converge to the global minimum, especially for non-convex optimization problems [34]. Additionally,

its performance is sensitive to the choice of learning rates, initialization, and hyperparameters [58]. These limitations of traditional SGD are overcome by ADAM optimizer.

ADAM

Adam [37] is an efficient stochastic optimization method that utilizes first-order gradients and minimal memory, computing adaptive learning rates for parameters based on estimates of gradient moments. This adaptive adjustment facilitates faster convergence and often yields superior performance compared to traditional gradient descent methods. It is designed to combine the advantages of the AdaGrad [25] optimization method, which works well with sparse gradients, and the RMSProp [59] optimization method which works well in online and non-stationary settings [37]. ADAM incorporates momentum, tracking the exponentially decaying average of past gradients to accelerate convergence. Additionally, it performs bias correction to stabilize the optimization process, particularly in the early stages of training. ADAM maintains the two moving averages of gradient, the first moment and second moment (Appendix A.1), which are then used to adaptively update learning rates for each parameter preventing it from being too high or too low. L2 regularization is also included to prevent overfitting by penalizing large parameter values, which are added to the loss function during optimization.

Due to its adaptive learning rate mechanism and efficiency in optimizing complex functions, the ADAM optimizer is commonly utilized in training PINNs. PINNs involve solving partial differential equations in high-dimensional spaces, presenting optimization challenges. ADAM's adaptive learning rate adjustment helps navigate the optimization landscape effectively, resulting in faster convergence and improved performance.

Limited-memory BFGS

The Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm [35] is a popular optimization technique used to minimize smooth, unconstrained objective functions. An unconstrained objective function refers to a function that does not have any constraints on the variables it operates on. This means that the variables can take any real values without being subject to any restrictions or limitations. The L-BFGS algorithm is a member of the quasi-Newton methods family, which approximates the BFGS algorithm [35] [63] while

utilizing a limited amount of computer memory [38]. It is widely utilized for parameter estimation tasks in machine learning [43] [16]. The primary objective of the algorithm is to minimize the scalar function $f(\mathbf{x})$ over unconstrained real-vector values $\mathbf{x}$, where f is a differentiable function.

The L-BFGS algorithm, like BFGS, uses an estimate of the inverse Hessian matrix to guide its search through variable space. However, while BFGS stores a dense approximation of the inverse Hessian, L-BFGS stores only a few vectors representing the approximation implicitly. This linear memory requirement makes L-BFGS ideal for optimization problems with numerous variables. Its quadratic convergence properties often result in a faster convergence rate compared to first-order methods like SGD or ADAM. However, LBFGS works best when optimization processes are well-initialized and may not be ideal for certain problems featuring an extensive parameter space or non-smooth loss functions, as it heavily relies on gradient computations and Hessian matrix approximations.

In linear elasticity PDEs, the objective function is typically unconstrained. PINNs seek to minimize the disparity between the predicted solution from the neural network and the governing PDEs as well as the boundary conditions. This disparity quantified as a loss function, measures the neural network's approximation of the true solution. Although there may be implicit physical constraints (such as displacement or stress continuity), these constraints are not directly included as equality or inequality constraints in the objective function. Instead, they are enforced implicitly through physics-based loss terms and boundary conditions during training. A prevalent strategy in PINN training involves initially employing ADAM for broader exploration and subsequently fine-tuning with LBFGS, a two-step approach that has demonstrated enhanced performance [33]. The selection of an optimization technique hinges on various factors, including the problem's characteristics, the scale of the problem space, the neural network's architecture, and the desired precision level. In this study, we employed either ADAM alone or a two-step approach involving ADAM and L-BFGS as needed for different problems.

2.2.3 PINNs Modeled using Soft and Hard Constraints

Boundary constraints for systems of PDEs can be implemented in two primary ways within the context of PINNs. In the soft constraint approach, the loss term arising from the boundary conditions is directly incorporated into the total loss function. PINNs are then trained to

minimize this total loss function, which includes contributions from both the PDE residual terms and the boundary condition terms. However, enforcing boundary conditions as soft constraints can pose challenges, as the neural networks need to adjust their parameters to satisfy these conditions effectively. Determining the appropriate weight for the boundary condition loss term can significantly impact the efficiency of the learning process. Currently, there is no established theory to guide the determination of this weight [22].

On the other hand, in the hard constraint approach, boundary conditions are enforced directly within the neural network architecture [40] [55]. This is achieved by transforming the output of the neural network using an equation such that it strictly satisfies the given boundary conditions. By enforcing boundary conditions as hard constraints, the need for iterative optimization to satisfy these conditions is eliminated, leading to reduced computational costs. This approach is versatile and can accommodate various types of boundary conditions, including Dirichlet, Neumann, Robin, or periodic boundary conditions [40]. However, it introduces additional complexity in the neural network architecture. Choosing the appropriate approach depends on the specific requirements and constraints of the problem at hand. A simple formulation of Dirichlet BC as a hard constraint is shown below. For a Dirichlet BC in form,

$$U_i(x) = g_0(x), \quad x \in \Gamma_D \tag{2.17}$$

where, $\Gamma_D \in \partial\Omega$ is a subset of boundary. A function can be formulated such that it describes the values on the boundary as [40],

$$\hat{U}_i(x;\theta_u) = g(x) + l(x)\mathcal{N}(x;\theta_u) \tag{2.18}$$

where, $g(x)$ is the continuous extension from Γ_D to Ω, $\mathcal{N}(x;\theta_u)$ is the network output, θ_u is the network parameters, and $l(x)$ is a function that only satisfies the boundary conditions as,

$$\begin{cases} l(x) = 0, & x \in \Gamma_D, \\ l(x) > 0, & x \in \Omega - \Gamma_D \end{cases} \tag{2.19}$$

The function $l(x)$ can be obtained analytically where Γ_D is a simple geometry. For example, if Γ_D is an interval such that, $\Gamma_D = \{a, b\}$, then an analytical function can be obtained such that it satisfies the condition in Equation (2.19) as, $l(x) = (x - a)(x - b)$. It can be difficult to obtain $l(x)$ analytically for complex domains and can be approximated using spline functions.

2.3 Linear Elasticity

Elasticity denotes the intrinsic property of a material to deform in response to external forces until it reaches a specific limit, known as the elastic limit. Beyond this limit, the material undergoes permanent deformation. This characteristic is fundamental to the behavior of structural materials, which exhibit varying degrees of elasticity. The linear elasticity equations are formulated based on the assumption of elastic behavior, wherein deformations are linearly proportional to applied forces, as governed by Hooke's Law. These equations serve as foundational principles in understanding the mechanical response of materials to external loading conditions.

The constitutive equation for the elastic behavior of a material is governed by the stress-strain relationship, commonly referred to as Hooke's Law of elasticity. This relationship is expressed mathematically as [32],

$$\sigma_{ij} = C_{ijkl}\, \epsilon_{kl} \tag{2.20}$$

where σ_{ij} represents the Cauchy stress tensor, which describes the distribution of internal forces within the material in response to external loads, C_{ijkl} denotes the fourth-order stiffness tensor, which characterizes the material's elastic properties and ϵ_{kl} is the strain tensor describing the deformation or change in the shape of the material caused by applied forces.

2.3.1 Governing Equations of Linear Elasticity

Linear elasticity problems are governed by a set of partial differential equations describing the equations of motion, strain-displacement relations, and constitutive equations, notably Hooke's Law. These problems can be formulated and solved for either static or dynamic states. In this thesis, our focus is on elastostatic problems, where inertia forces are neglected, and only static equilibrium is considered. The key equations that collectively define the behavior of linear elastic materials under static loading conditions are given below. They are initially presented in Einstein's summation convention and subsequently elaborated into engineering notations for three-dimensional problems in Cartesian coordinates.

1. Equation of motion for static equilibrium [32]

$$\sigma_{ji,j} + F_i = 0 \tag{2.21}$$

In engineering notation,

$$\frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} + F_x = 0 \tag{2.22}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} + F_y = 0 \tag{2.23}$$

$$\frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + F_z = 0 \tag{2.24}$$

where σ_{ji} is the Cauchy stress tensor and the subscript j is a shorthand for $\partial(\bullet)/\partial x_j$, F_i is the body force.

2. Strain-displacement relation [32]

$$\epsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right) \tag{2.25}$$

In engineering notation,

$$\epsilon_{xx} = \frac{\partial u_x}{\partial x} \tag{2.26}$$

$$\epsilon_{yy} = \frac{\partial u_y}{\partial y} \tag{2.27}$$

$$\epsilon_{zz} = \frac{\partial u_z}{\partial z} \tag{2.28}$$

$$\epsilon_{xy} = \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right) \tag{2.29}$$

$$\epsilon_{yz} = \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}\right) \tag{2.30}$$

$$\epsilon_{zx} = \frac{1}{2}\left(\frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z}\right) \tag{2.31}$$

where ϵ_{ij} is the strain tensor and u_i are the displacements.

3. Hooke's law (Equation 2.20) or Constitutive equations can be expressed using Lamé's parameters and are written as [32]

$$\sigma_{ij} = \lambda \delta_{ij} \epsilon_{kk} + 2\mu \epsilon_{ij} \tag{2.32}$$

In engineering notation,

$$\sigma_{xx} = 2\mu\epsilon_{xx} + \lambda\left(\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}\right) \tag{2.33}$$

$$\sigma_{yy} = 2\mu\epsilon_{yy} + \lambda\left(\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}\right) \tag{2.34}$$

$$\sigma_{zz} = 2\mu\epsilon_{zz} + \lambda\left(\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}\right) \tag{2.35}$$

$$\sigma_{xy} = 2\mu\epsilon_{xy} \tag{2.36}$$

$$\sigma_{yz} = 2\mu\epsilon_{yz} \tag{2.37}$$

$$\sigma_{zx} = 2\mu\epsilon_{zx} \tag{2.38}$$

where λ is Lamé's first parameter, δ_{ij} is the Kronecker delta and μ is Lamé's second parameter or shear modulus. In an isotropic material, the shear stresses exhibit symmetric behavior in the stress tensor. Specifically, this symmetry implies that σ_{xy} = σ_{yx}, σ_{yz} = σ_{zy}, and σ_{zx} = σ_{xz}. These relationships reflect the uniformity of material properties across different orientations and are assumed throughout this document for consistency in analysis.

Typically, it is assumed that the body forces F_i are known. Thus, the solution sought from the fifteen equations listed here pertains to the six stresses σ_{ij}, the six strains ϵ_{ij}, and the three displacements u_i. The displacement conditions and traction conditions on the boundary can be prescribed in different ways to obtain the solution to the problem as discussed in the following conditions [32]:

1. Displacement condition on boundary $\partial\Omega$

$$u_i = u_{i_0}(x) \quad \text{on } \Omega \tag{2.39}$$

2. Traction condition on boundary $\partial\Omega$

$$\sigma_i \cdot n_i = T_i(x) \quad \text{on } \Omega \tag{2.40}$$

3. Displacement condition on boundary $\partial\Omega_1$ and traction condition on boundary $\partial\Omega_2$, where $\partial\Omega_1 \cup \partial\Omega_2 \in \partial\Omega$

$$u_i = u_{i_0}(x) \quad \text{on } \Omega_1 \tag{2.41}$$

$$\sigma_i \cdot n_i = T_i(x) \quad \text{on } \Omega_2 \tag{2.42}$$

where $u_{i_0}(x)$ is the known displacement vector at every point on the boundary, n_i is the unit outward normal vector to the boundary at position x, $T_i(x)$ is the traction vector field, representing the distribution of forces or stresses acting on the boundary, the dot product $\sigma_i \cdot n_i$ represents the traction (force per unit area) acting on the boundary in the direction of the normal vector.

In summary, displacement conditions and traction conditions are boundary conditions that govern the behavior of linear elastic materials at the boundaries of the domain. Displacement conditions prescribe the displacements of the material, while traction conditions specify the distribution of forces or stresses acting on the material. These boundary conditions play a fundamental role in defining the overall behavior and response of the material under various loading conditions in linear elasticity problems.

2.3.2 Traction Boundary Conditions

The surface force per unit area is called traction [32], denoted by vector t. As introduced in section 2.3.1, the traction conditions on any arbitrary boundary can be defined by dot product of stress tensor σ and a unit normal vector n, as $\sigma \cdot n = t$. This can be written as [32],

$$\sigma_{ij} \cdot n_j = t_i \tag{2.43}$$

For a three-dimensional problem, Equation (2.43) can be expanded as,

$$\sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z = t_x \tag{2.44}$$

$$\sigma_{yx}n_x + \sigma_{yy}n_y + \sigma_{yz}n_z = t_y \tag{2.45}$$

$$\sigma_{zx}n_x + \sigma_{zy}n_y + \sigma_{zz}n_z = t_z \tag{2.46}$$

If the surface is planar, the normal vector would be only in one direction, for example, in the x-direction the normal vector would be $(\mathbf{1\ 0\ 0})$, then Equation (2.44), (2.45) and (2.46) can be simplified as,

$$\sigma_{xx} = t_x, \quad \sigma_{yx} = t_y, \quad \sigma_{zx} = t_z \tag{2.47}$$

where t_x, t_y, t_z are the components of the traction vector, σ_{xx} is the stress in the normal direction often called the normal stress and σ_{yx} and σ_{zx} are the shear stresses.

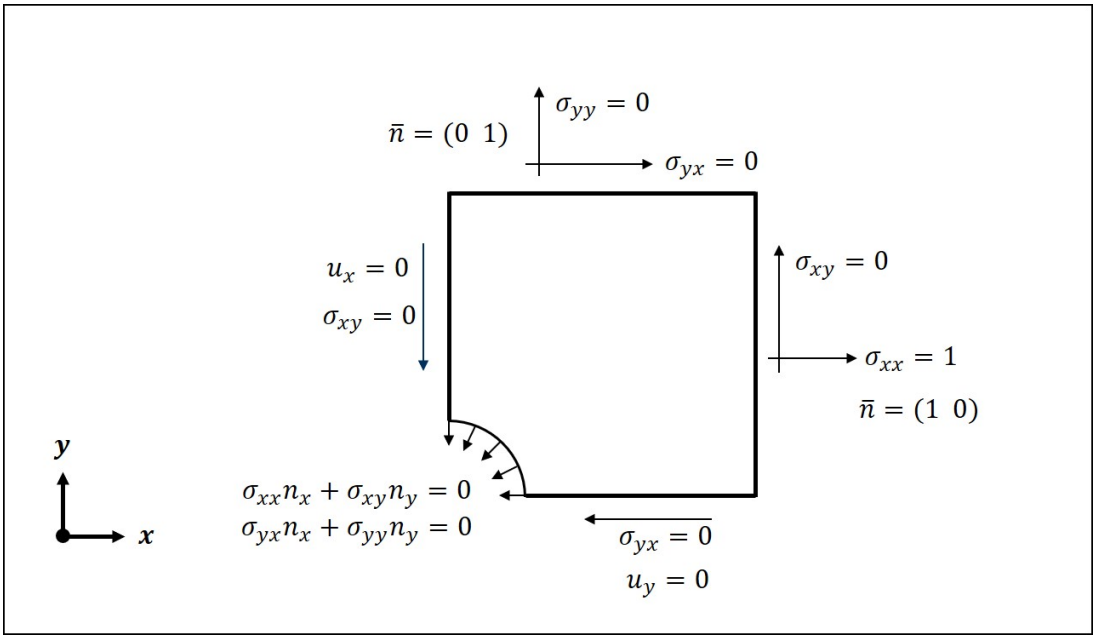


Figure 2.6: An illustration of displacement and traction conditions on a plane and a curved surface of 2D plate

If a force is applied perpendicular to a plane surface, the traction boundary condition would specify the magnitude and direction of the traction vector at each point along that surface. Similarly, if no external forces are acting on the surface, the traction boundary condition would indicate a zero traction vector. An illustrative example of a simple two-dimensional plate can be seen in figure 2.6, the displacement conditions are applied on the left and bottom boundary, the traction force of 1 N/m^2 is applied on the right boundary and the rest of the boundaries are traction-free.

3 Methodology

The preceding chapter covered the basics of neural network architecture, PINN architecture, different optimization methods, and the governing equations of linear elasticity. In this chapter, we will delve into various methods employed to train the PINN model for linear elasticity equations and the usage of different libraries and software packages.

3.1 Implementation of PINNs for Linear Elasticity

In the section 2.3, the governing equations of linear elasticity were introduced. To formulate the loss function, the residuals of these PDEs, along with the initial and boundary conditions, are taken into account. Various methods exist for evaluating this loss function, which measures the disparity between the predicted and actual solutions. Among these methods, a prevalent and standard approach involves utilizing the Mean Squared Error (MSE) formulation. In its fundamental expression, MSE represents the mean of the squared differences between the reference and predicted values. Mathematically, it can be represented as,

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 \tag{3.1}$$

where, y_i is the reference solution, $\hat{y}_i$ is the predicted solution and N is the number of collocation points.

The loss function associated with the PDEs governing linear elasticity can be expressed as,

$$\begin{aligned}
\mathcal{L}_{\mathcal{F}}(\theta) = \frac{1}{N}\sum_{i}^{N}\Big(& \parallel \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{yx}}{\partial y} + \frac{\partial \sigma_{zx}}{\partial z} + F_x \parallel^2 \\
& + \parallel \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y} + \frac{\partial \sigma_{zy}}{\partial z} + F_y \parallel^2 \\
& + \parallel \frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{yz}}{\partial y} + \frac{\partial \sigma_{zz}}{\partial z} + F_z \parallel^2 \\
& + \parallel \sigma_{xx} - 2\mu\epsilon_{xx} + \lambda\left(\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}\right) \parallel^2 \\
& + \parallel \sigma_{yy} - 2\mu\epsilon_{yy} + \lambda\left(\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}\right) \parallel^2 \\
& + \parallel \sigma_{zz} - 2\mu\epsilon_{zz} + \lambda\left(\epsilon_{xx} + \epsilon_{yy} + \epsilon_{zz}\right) \parallel^2 \\
& + \parallel \sigma_{xy} - 2\mu\epsilon_{xy} \parallel^2 + \parallel \sigma_{yz} - 2\mu\epsilon_{yz} \parallel^2 + \parallel \sigma_{zx} - 2\mu\epsilon_{zx} \parallel^2
\end{aligned} \tag{3.2}$$

Similarly, the loss function related to displacement boundary conditions and traction conditions can be expressed as follows,

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{N}\sum_{i}^{N}\left(\parallel u_i^N - u_{i_0}(x) \parallel^2 + \parallel \sigma_i^N \cdot n_i - T_i(x) \parallel^2 \right) \tag{3.3}$$

where, u_i^N is the output of displacement from neural network, $u_{i_0}(x)$ is the known displacement vector, σ_i^N is the output of stress from neural network and $T_i(x)$ is the known traction vector.

Each term in the loss function is assigned specific weights to balance their contributions and guide the training process effectively. Balancing the weights in the loss function is crucial to ensure that each term influences the model training appropriately. This approach helps in prioritizing certain aspects of the prediction, such as accurately capturing boundary conditions or minimizing errors in specific output parameters. In practice, boundary losses are prioritized over PDE losses by assigning them higher weights. This approach ensures that the neural network accurately captures and respects the given constraints, guiding the model towards more robust and accurate predictions within the specified domain.

Additionally, the output from the final layer of the model is transformed to accommodate hard constraints and provide a more refined initial guess for the PINN model. For example, consider the problem configuration illustrated in figure 2.6. To apply hard constraints effectively, we use the following formulations: $u_x^T = x \cdot u_x^N$ for the displacement in x-direction; $u_y^T = y \cdot u_x^N$ for the displacement in y-direction. These formulations satisfy Equation (2.19), ensuring that the transformed outputs u_x^T and u_y^T incorporate the necessary constraints for the problem configuration. The transformed output is scaled according to the order of magnitude of each output variable. For instance, if u_x has an order of magnitude of 0.1, u_y of 0.01, and σ_{xx} of 10, then the transformation is applied as follows: $u_x^T = x \cdot u_x^N \times 0.1$, $u_y^T = y \cdot u_y^N \times 0.01$ and $\sigma_{xx}^T = \sigma_{xx}^N \times 10$. Here, $(\cdot)^N$ represents the output from the neural network output layer, and $(\cdot)^T$ denotes the transformed output used for calculating the losses within the PINN framework. This scaling ensures that the transformed outputs maintain appropriate scales and magnitudes. The incorporation of hard constraints and scaled transformations contributes to enhancing the model's accuracy and convergence, particularly in scenarios involving complex structural behaviors.

In the previous sections 2.1.2 and 2.2, we discussed the architecture of an MLP and its augmentation with physics-informed networks for loss optimization and training. Recognizing the limitations of traditional MLPs, particularly in handling multiple inputs and outputs efficiently, a Multi-Feedforward Neural Network (MFNN) [40] [55] was employed. The MFNN structure consists of individual MLPs for each output, as depicted in figure 3.1. Unlike a conventional MLP where multiple outputs are connected directly to the hidden layers, in an MFNN, each output is connected to a separate MLP. This architecture facilitates independent convergence of each output, thereby enhancing the model's convergence rate and overall performance. In this thesis, focusing on linear elasticity equations, we handle 5 outputs (2 displacements and 3 stresses) in a two-dimensional scenario, and 9 outputs (3 displacements and 6 stresses) in a three-dimensional case. As a result, we utilize 5 MLPs and 9 MLPs for each respective scenario. It's worth noting that while MFNN yields superior convergence and overall performance, it necessitates additional memory for deploying multiple MLPs. A diagram illustrating MFNN for two-dimensional linear elasticity equations is depicted in figure 3.1. In figure 3.2, 5 independent networks are employed for values of interest, u_x, u_y, σ_{xx}, σ_{yy}, σ_{xy}. Each network has x, and y as input features. The network is updated to 9 independent networks in a three-dimensional case with x, y, and z as input features.

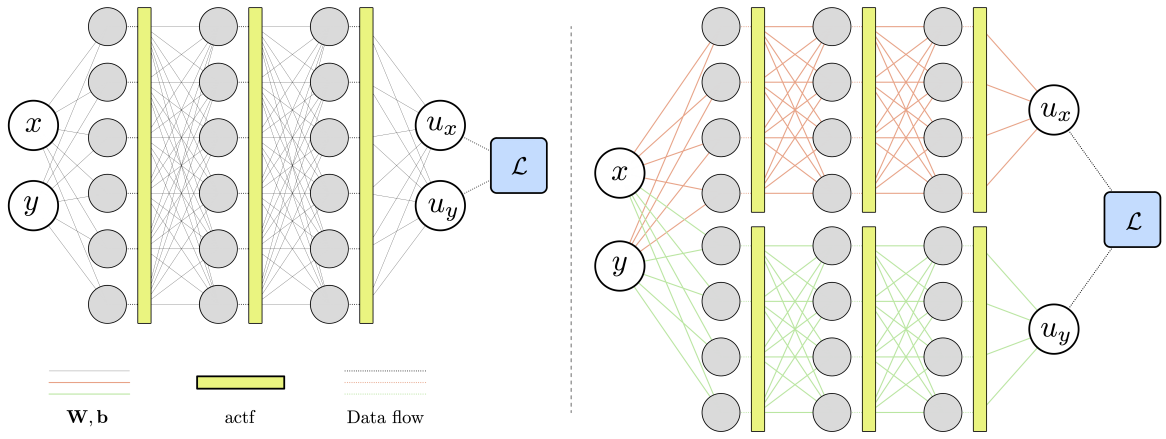


Figure 3.1: The figure represents the neural network architecture [55] with u_x and u_y as the output of a single MLP network (left) and NN architecture with u_x and u_y as the output of two independent MLP network with different parameters (right)

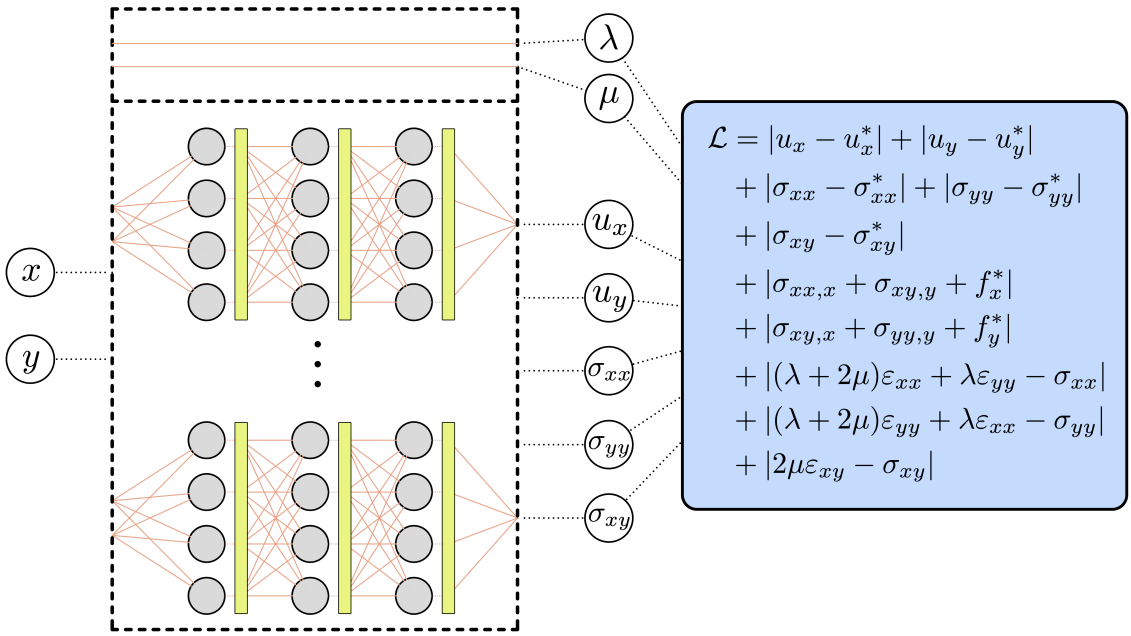


Figure 3.2: The figure represents the neural network architecture [55] chosen for this study in a two-dimensional case.

3.2 Usage of Different Libraries and Software

Various software packages are available for training the PINN model, including DeepXDE, NVIDIA Modulus, PYDENs, and NeuroDiff.EQ. These libraries employ feed-forward neural networks as the foundational architecture. They utilize automatic differentiation techniques to analytically compute derivatives, which are essential for determining the loss function during training.

DeepXDE: A Deep Learning Library

For this thesis, the DeepXDE [39] which is a library for scientific machine learning and physics-informed learning has been utilized. It offers both Feedforward Neural Network (FNN) and Multi-Feedforward Neural Network (MFNN) architectures, referred to as PFNN in DeepXDE. This versatile package supports five tensor libraries as backends: TensorFlow 1.x (tensorflow.compat.v1 in TensorFlow 2.x), TensorFlow 2.x, PyTorch, JAX, and PaddlePaddle, and also provides a range of optimizers and initializers to choose from. Users have the flexibility to define custom loss functions and optimizers tailored to their specific problem def-

initions. While DeepXDE includes predefined functions for generating simple geometries, more complex geometries require the construction of custom geometry classes or the utilization of point clouds. Additionally, users can define new neural network architectures by following the constructor framework provided by DeepXDE. A general approach for defining the problem using DeepXDE modules is shown in figure 3.3 and is as follows [39]:

1. Specify the computational domain using the **`geometry`** module.
2. Specify the PDE using the grammar of **`Tensorflow`**.
3. Specify the boundary and initial conditions.
4. Combine the geometry, PDE, and boundary/initial conditions together into **`data.PDE`** or **`data.TimePDE`** for time-independent problems or for time-dependent problems, respectively. To specify training data, we can either set the specific point locations or only set the number of points, and then DeepXDE will sample the required number of points on a grid or randomly.
5. Construct a neural network using the **`maps`** module.
6. Define a **`Model`** by combining the PDE problem in step 4 and the neural net in step 5.
7. Call **`Model.compile`** to set the optimization hyperparameters, such as optimizer and learning rate. The weights in Equation (2.14) can be set here by **`loss_weights`**.
8. Call **`Model.train`** to train the network from random initialization or a pre-trained model using the argument **`model_restore_path`**. It is extremely flexible to monitor and modify the training behavior using **`callbacks`**.
9. Call **`Model.predict`** to predict the PDE solution at different locations.

Computer-Aided Engineering (CAE) Softwares: Gmsh and Abaqus

Gmsh [30] is an open-source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light, and user-friendly meshing tool with parametric input and flexible visualization capabilities. Gmsh is built around four modules (geometry, mesh, solver, and post-processing), which can be controlled with the graphical user interface, from the command line, using text files written in Gmsh's scripting language (.geo files), or through the C++, C, Python, Julia, and Fortran application program-

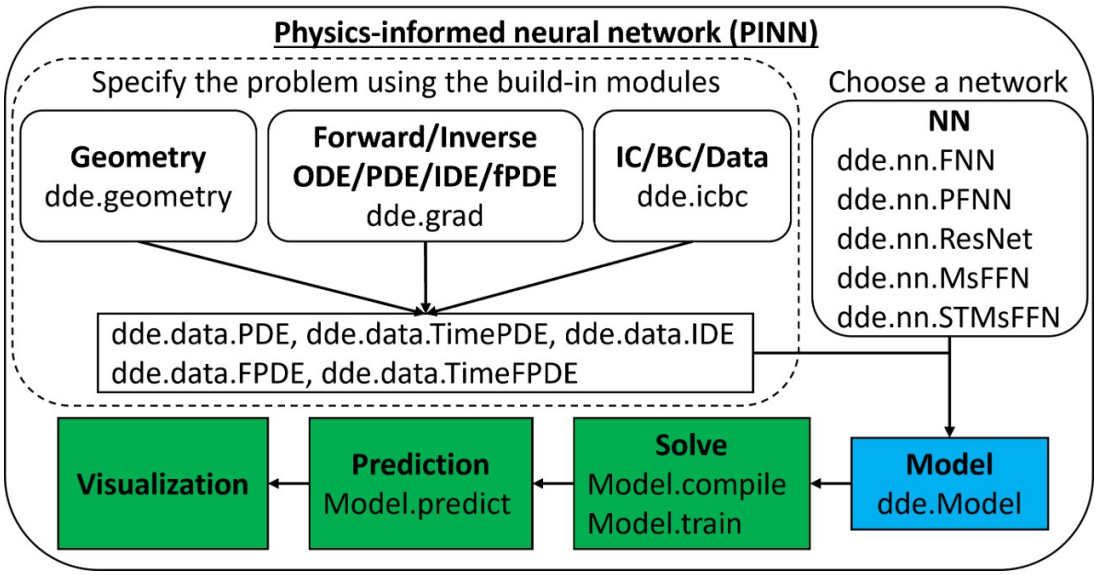


Figure 3.3: A representation of general approach to define the problem using DeepXDE [39]

ming interface.

As previously mentioned, while the DeepXDE library offers built-in geometry modules for defining simple shapes, more complex 2D and 3D geometries with intricate boundaries can pose challenges. In such cases, the `PointCloud` function in DeepXDE becomes valuable, allowing the definition of points in space. These points can be imported from Gmsh, a tool that offers flexibility in creating complex geometries. Gmsh allows for mesh refinement in specific regions, enabling the desired distribution of points in the domain and boundaries. Additionally, Gmsh facilitates the definition of physical boundaries on its CAD model, simplifying access to points on surface boundaries or within different regions.

Abaqus is a powerful finite element analysis commercial software used for simulating and analyzing complex structural, thermal, and multiphysics problems. Abaqus offers a wide range of capabilities for modeling, meshing, simulating, and visualizing mechanical behavior and performance. Apart from its user-friendly GUI, it also offers Python scripting capabilities, allowing users to automate simulations, customize workflows, and extend the software's functionality. In this project, the accuracy of the PINN model was verified by comparing its results with those obtained from Abaqus software. Additionally, Abaqus Python scripting was employed to import force data from an external CFD tool.

AeroSandBox Tool

One of the primary aims of this thesis is to develop a PINN model for a complex 3D airfoil, which serves as a simplified representation of an aircraft wing, depicted in figure 3.4. The objective was to demonstrate that the PINN model can accurately predict solutions to linear elasticity problems under realistic lift forces generated by high pressure beneath the wing as air flows over it. To achieve this, we employed the widely-used AeroSandBox tool, known for its reliability in various aerodynamic calculations. AeroSandBox [51] is an open-source Python-based tool. In our approach, we captured the lift forces acting on the airfoil and calculated the pressure on each surface to apply as traction force in the vertical direction during both PINN model training and FEM simulation. In the PINN model, the traction force was incorporated as traction boundary conditions on the bottom surface. To compute the lift forces, we utilized the Vortex Lattice Method (VLM) provided by the AeroSandBox tool, which will be elaborated upon in the next section. It's worth noting that the mesh of the airfoil differs between AeroSandBox and FEM software, necessitating the mapping of traction values from the coarse mesh of AeroSandBox to the finer mesh of FEM software. The methodology for mapping pressure values from the coarse mesh to the finer mesh will be discussed in detail in the subsequent section.

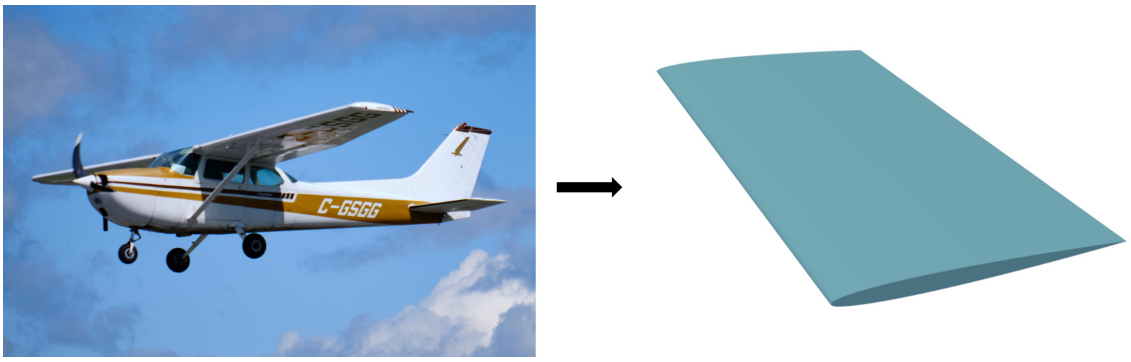


Figure 3.4: The figure represents the simplified version of an aircraft wing (right) and the actual aircraft [3] (left)

3.3 Mapping Forces from AeroSandBox Tool

In this section, we will delve into the methodology employed to extract forces from the AeroSandbox tool and its subsequent mapping from a coarse mesh to a fine mesh for application to the FEM model and PINN model. A brief overview of the vortex lattice method utilized for calculating lift forces is also provided.

3.3.1 Vortex Lattice Method (VLM)

The Vortex Lattice Method (VLM) [1] [23] is a numerical method used in CFD, in the early stages of the aircraft design to model the aircraft wing, as an infinitely thin sheet of discrete vortices to compute lift and induced drag. The influence of the thickness and viscosity is neglected. The Vortex Lattice Method begins by selecting a thin surface, typically positioned along the camber line of the airfoil. All computations are conducted on this surface, where the total forces are presumed to act. This thin surface is then discretized into a lattice of panels, with each panel representing a small segment of the surface. These panels are commonly assumed to be flat. The arrangement of panels in the lattice is depicted in figure 3.5.

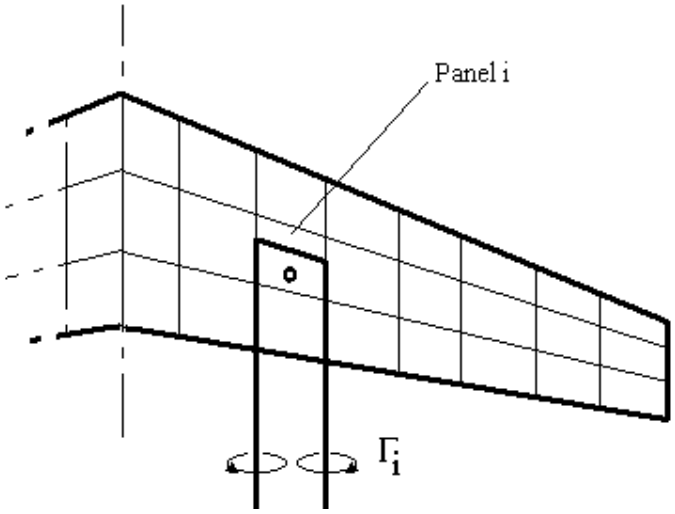


Figure 3.5: A thin surface of aircraft wing discretized into the lattice of panels [1]

Within each panel, the bound vortex, Γ_i is positioned at the quarter chord (represented by three lines), while the control point is located at the third quarter chord (represented by point), as illustrated in figure 3.5. These positions are calculated for every panel using the

coordinates of all four vertices of the panel. The boundary conditions are enforced at these control points, where the velocities (including induced velocity and free stream velocity) are set to zero. The free stream velocity corresponds to the aircraft's velocity, while the induced velocity is determined using the Biot-Savart Law [5]. Formulating the Biot-Savart Law for each horseshoe vortex on every panel results in a system of equations. Solving this system yields the vortex strength for each panel. Subsequently, using the induced velocity, perturbation velocity, and vortex strength, the lift forces are computed for each panel. Summing these lift forces provides the total lift. In our scenario, it is necessary to capture the lift forces on each panel and apply them as traction conditions. To achieve this, the traction force, T_i on each panel is obtained by calculating the lift force L_i per unit area A_i for every panel ($T_i = L_i/A_i$).

3.3.2 Implentation of Mapping Method

In this subsection, we will outline the mapping technique employed to transfer traction values from the AeroSandBox mesh to the FEM mesh. Given that the mesh from the AeroSandBox tool is coarser than the FEM mesh, we'll refer to the AeroSandBox mesh as the coarse mesh and the FEM mesh as the fine mesh.

Initially, the 3D bottom surface of the airfoil is projected onto a 2D surface, disregarding the thickness in the y-direction to obtain a 2D mesh. The objective is to map values from the coarse mesh to this fine mesh. To achieve this, the centroid, C_f of every face of the fine mesh is computed. Subsequently, the bounding vertices (x_{min}, x_{max}, y_{min}, y_{max}) of the faces of the coarse mesh are determined, and an iterative process is undertaken for all faces of the fine mesh to ascertain whether the centroid, C_f of each fine mesh face lies within the bounding vertices of the corresponding coarse mesh face. If this condition is met, the face of the fine mesh is considered to lie inside the face of the coarse mesh. This procedure is repeated for all other fine mesh faces. Each face of the fine mesh is then assigned a traction value T_f identical to that of the coarse mesh face. Finally, the traction force value of each face of the fine mesh is divided by the total number of fine mesh faces within the coarse mesh's face. This ensures that the net traction force in that area remains consistent.

While this method may not offer pinpoint accuracy due to some areas of the fine mesh faces not completely lying within the coarse mesh face, it presents a straightforward imple-

mentation and is sufficiently accurate for our purposes. The pseudocode of this procedure is depicted in Algorithm 1. The traction forces obtained for each fine mesh face are applied to the corresponding faces of the fine mesh within the software using Abaqus scripting. Additionally, these traction values serve as boundary conditions in the PINN model, applied to the boundary points of the bottom surface.

Algorithm 1 Assign Traction Values to Fine mesh faces inside Coarse mesh faces

Require: F_c^*, F_f^*, X_c, X_f, T_c
Ensure: Updated T_{f_j}
for $i \leftarrow 0$ to $|F_c^* - 1|$, $F_{c_i} \in F_c^*$ **do**

    Calculate bounding box of F_{c_i}:
    $x_{c_{min}},\ x_{c_{max}} \leftarrow \min(x_{c_1}, x_{c_2}, x_{c_3}, x_{c_4}),\ \max(x_{c_1}, x_{c_2}, x_{c_3}, x_{c_4})$
    $y_{c_{min}},\ y_{c_{max}} \leftarrow \min(y_{c_1}, y_{c_2}, y_{c_3}, y_{c_4}),\ \max(y_{c_1}, y_{c_2}, y_{c_3}, y_{c_4})$

    for $j \leftarrow 0$ to $|F_f^* - 1|$, $F_{f_j} \in F_f^*$ **do**

        Calculate centroid of F_{f_j}:
        $C_{f_x} \leftarrow (x_{f_1} + x_{f_2} + x_{f_3} + x_{f_4})/4$ ▷ x-coordinate of centroid
        $C_{f_y} \leftarrow (x_{f_1} + x_{f_2} + x_{f_3} + x_{f_4})/4$ ▷ y-coordinate of centroid

        if $(x_{c_{min}} \leq C_{f_x} \leq x_{c_{max}})$ **and** $(y_{c_{min}} \leq C_{f_y} \leq y_{c_{max}})$ **then**
            $T_{f_j} \leftarrow T_{c_i}$ ▷ Update traction value of F_{f_j}

        end if
    end for
end for

Initialize $\mathbf{N} \leftarrow 0$
for $i \leftarrow 0$ to $|T_f - 1|$, $v_i \in T_f$ **do** ▷ Loop over each traction value
    $count = n(v_i \in T_f)$ ▷ count number of repeated traction values
    $\mathbf{N}_i =$ count ▷ assign the count value ith element of $\mathbf{N}$
end for

$T_{f_{\text{mapped}}} = T_f/\mathbf{N}$ ▷ divide the pressure values

An illustrative example of this mapping method is depicted in figure 3.6 and 3.7. In this illustration, T_1, T_2, T_3, and T_4 represent the traction forces on the faces of the coarse mesh. Let's consider the first face with the value T_1. Upon examination, it is evident that the centroid (depicted by black dots) of 12 fine mesh faces lies within the area of the coarse mesh face. Consequently, the traction value on each fine mesh face is calculated by dividing the traction value of the coarse mesh face by the number of fine mesh faces it encompasses, yielding T_1/12. This procedure is repeated iteratively for all faces to ensure accurate mapping of the traction values.

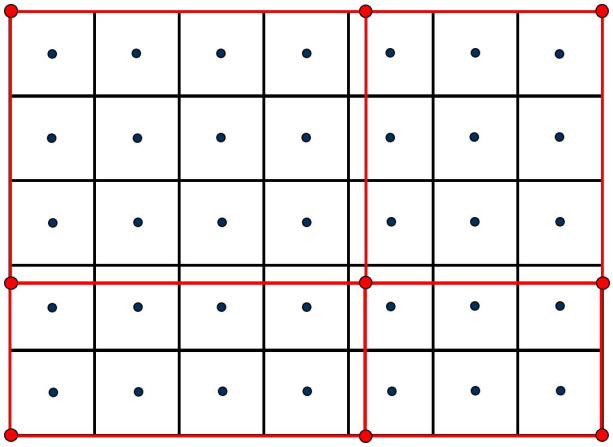


Figure 3.6: Depiction of coarse mesh (red) on fine mesh (black). The vertices of the coarse mesh are represented with red dots and the centroid of faces of fine mesh are represented with black dots

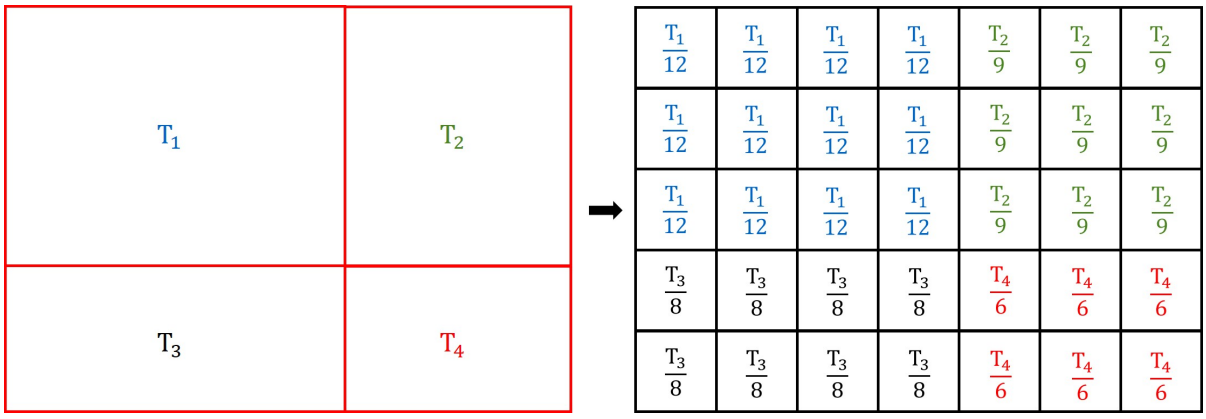


Figure 3.7: Mapping of traction values from coarse mesh to fine mesh

4 Results and Validation

In this chapter, we will delve into the solution predicted by the PINN model and conduct a thorough examination, comparing them with the reference solution obtained from Abaqus. Initially, we will focus on assessing results obtained for two-dimensional geometries across various studies. Following this, we will explore the intricacies encountered during the training of the three-dimensional PINN model, shedding light on the challenges related to training and convergence. Lastly, we will explore the prospects of the PINN model and potential solutions to the issues identified in this study.

4.1 Evaluation and Validation of Two Dimensional Geometries

4.1.1 Pure Bending Beam

In this study case, we present a simple problem of a pure bending beam. The beam is subjected to a bending moment, M = 1/12 Nm, implemented as traction force, $\bar{t}_M(y)$ = 1000y. The dimensions and the bending moment value are taken as reference from research paper [17] for initial validation. Given the symmetry of the problem configuration, we can leverage this symmetry in the reduced model to decrease computation time and improve performance. The reduced model is represented in figure 4.1.

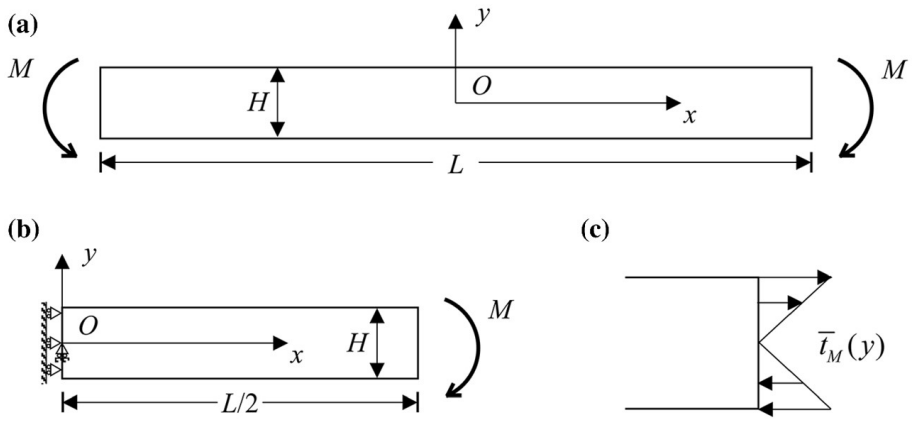


Figure 4.1: (a)The configuration of pure bending beam problem, (b) A reduced half model due to symmetry, (c) The bending moment implemented as traction force [17]

Here the length of the beam, L = 2 m, and height, H = 0.2 m, Young's modulus, E = 1000 Pa, and the Poisson's ratio, ν = 0.3. The displacement boundary conditions on the beam are

as follows,

$$u_x(0, y) = 0 \ \ , u_y(0, 0) = 0 \tag{4.1}$$

The PINN model is trained for 12,000 iterations with 1500 training points, 500 points on the boundary, and 1000 points randomly distributed inside the domain. The predicted output is compared with the analytical solution which is given as [17] [46],

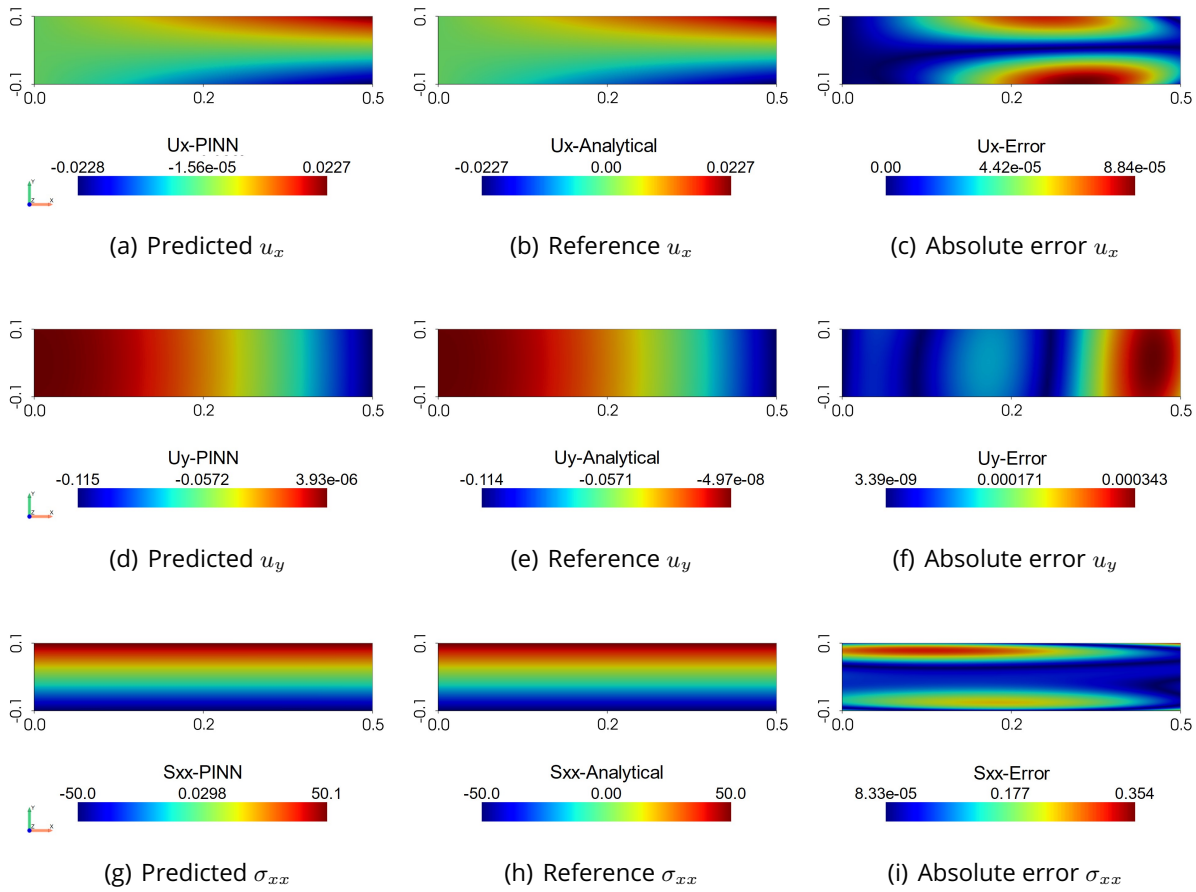
$$U_x = \frac{3M(2\mu + \lambda)xy}{\mu(\mu + \lambda)LH^3} \tag{4.2}$$

$$U_y = -\frac{3M(2\mu + \lambda)}{\mu(\mu + \lambda)LH^3}\left[(2\mu + \lambda)x^2 + \lambda y^2)\right] \tag{4.3}$$

The predicted results are accurate with mean absolute error for displacement in x-direction, e_{u_x} = 3.14×10^{-5}, displacement in y-direction, e_{u_y} = 1.12×10^{-4}, and normal stress in x-direction, $e_{\sigma_{xx}}$ = 1.07×10^{-1}. The analytical solution of σ_{yy} and σ_{xy} is zero hence they are omitted here. The comparison of the predicted solution and the reference solution is shown in figure 4.2.

The figure displaying the absolute error highlights that errors are more pronounced near or on the boundary where the output values reach their maximum. For instance, in figure 4.2(c), noticeable errors occur near the top and bottom boundaries, indicating that the PINN model successfully predicted the extreme values located at the corners of the right boundary, but further training is necessary to enhance the model's capability to accurately capture patterns near these boundary regions. Additionally, it is noticeable that the error is more pronounced on the bottom boundary compared to the top, which could be attributed to factors such as the distribution of points within the domain and on the boundaries, as well as the initialization of neural network parameters. It is important to note that achieving convergence in PINN training depends on various factors including the distribution and quantity of training points, the complexity of the neural network architecture (number of neurons and layers), and the initialization of weights (e.g., Glorot normal initialization). It is also worth mentioning that training the PINN model multiple times may not lead to identical solutions each time, although the outcomes should be comparable.

A similar observation applies to the error depicted in figure 4.2(f), where the maximum error occurs near the boundary corresponding to the maximum displacement value. Likewise, in figure 4.2(i), the error is more pronounced on the top boundary compared to the bottom boundary.



(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

(g) Predicted σ_{xx} (h) Reference σ_{xx} (i) Absolute error σ_{xx}

Figure 4.2: Pure bending beam problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of u_x, u_y, and σ_{xx} respectively

4.1.2 Thin Perforated Plate

In this subsection, a simple two-dimensional perforated strip plate with a hole in the center is subjected to plane stress on the boundaries as shown in figure 4.3(a). The problem definition can be simplified due presence of symmetry and formulated as a quarter plate subjected to stress on the right boundary keeping the left edge and bottom edge (edges that are cut through the plane of symmetry). This is shown in figure 4.3(b). This study analyzes different implementations of constraints and the computational efficiency of different hardware structures. Firstly, the displacement conditions as given as hard constraints, and the accuracy of the results is validated with an analytical solution. Further, it is compared with modeling the displacement boundary conditions as soft constraints in the PINN model, similar to traction boundary constraints.

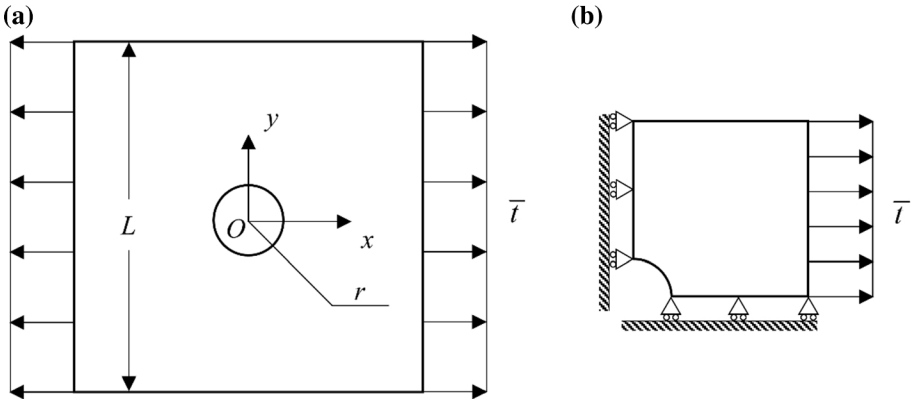


Figure 4.3: (a) A perforated strip plate subjected to uniform traction, (b) Simplified quarter plate model [17]. The material parameters are: E = 5 Pa, $\nu = 0.3$

Validation of the Predicted Solution

The thin perforated plate with the dimensions, L = 2 m and radius, r = 0.2 m (the mentioned dimension are chosen for initial validation from paper [17]), was subjected to uniform traction, $\bar{t}$ = 1 N/m^2. The displacement and traction conditions for this particular problem are shown in figure 2.6. The outcomes were achieved by training the PINN model using a dataset of 5000 points, comprising 4000 points located within the domain and 1000 points on the boundary. The reference solution utilized for validation was obtained from FEM software. The mesh of the FEM model consists of 11,471 nodes and 11,449 elements to obtain accurate results. The PINN model obtained the accurate results with a mean absolute error for outputs, $e_{u_x} = 9.68 \times 10^{-4}$, $e_{u_y} = 9.12 \times 10^{-4}$, $e_{\sigma_{xx}} = 5.82 \times 10^{-3}$, $e_{\sigma_{yy}} = 5.51 \times 10^{-3}$ and $e_{\sigma_{xy}} = 2.95 \times 10^{-3}$, after running the training process for 30,000 iterations (5000 iterations with ADAM and 25,000 iterations with LBFGS).

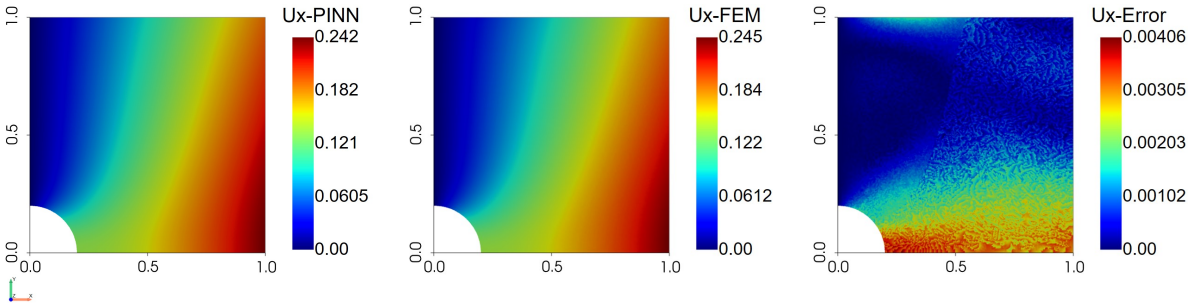
Table 4.1: Computation details for PINNs and FEM model

Model	Training time (s)	Simulation time (s)	Prediction time (s)
PINN	4144.27	-	0.0934
ABAQUS	-	0.74	-

The computational details for both models are presented in table 4.1. It is evident that training the PINN model is computationally expensive compared to the FEM software; however, the prediction time for the PINN model is significantly lower. This underscores the importance of enhancing the PINN model to reduce its computational cost and make it

more competitive with FEM software for efficient and promising engineering simulations and analyses.

The observations from figure 4.4 and 4.5 highlight a disparity in prediction accuracy between displacements and stresses within the PINN model. The displacements exhibit relatively small errors, whereas stresses, being more nonlinear, display larger errors, indicating the greater complexity in learning and predicting stress fields compared to displacements.

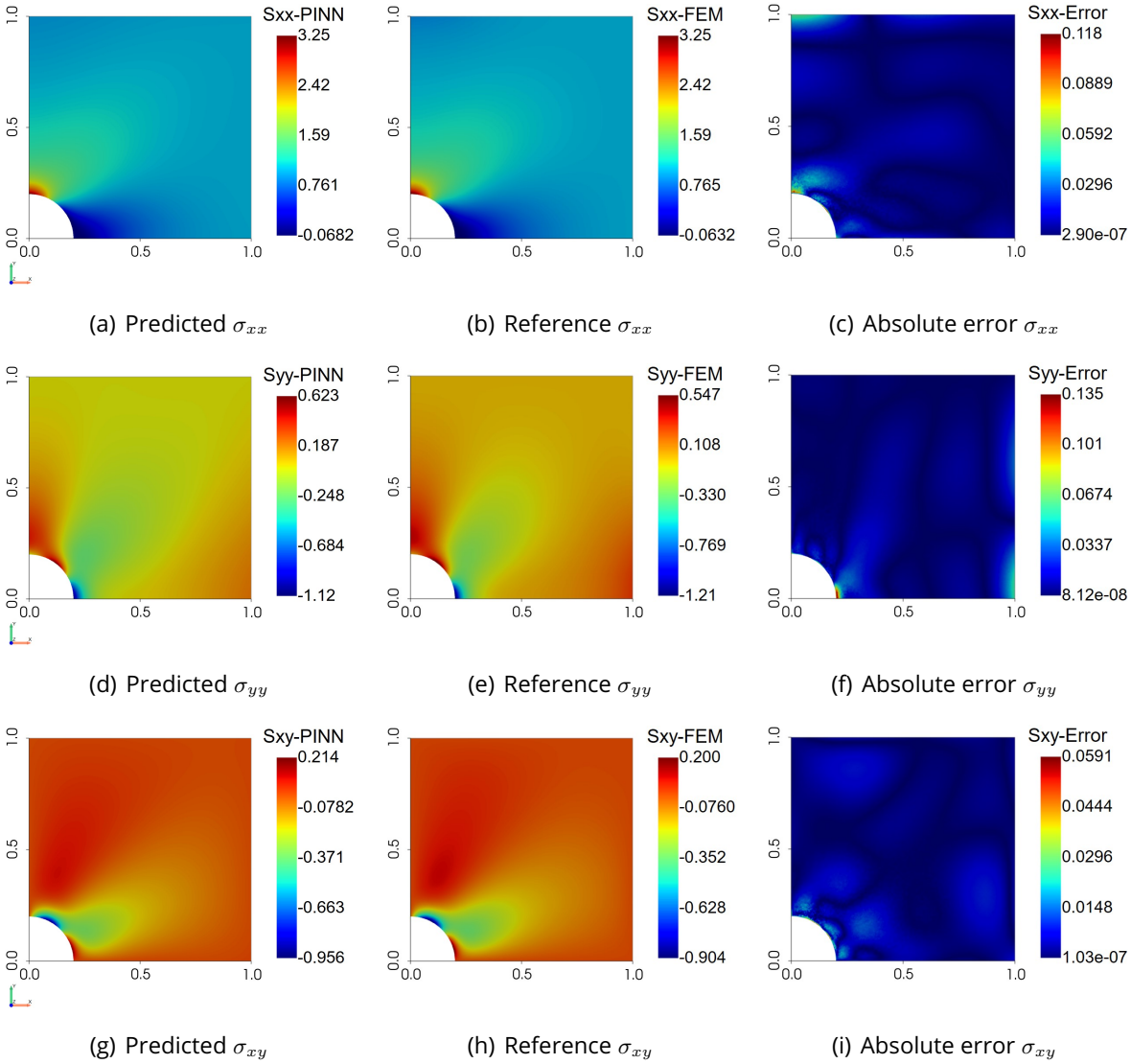


(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

Figure 4.4: Plane stress problem: (a), (d) are the predicted solutions; (b), (e) are the reference solutions; (c), (f) are the absolute differences between the reference and predicted solutions, for the value of u_x and u_y respectively

The deviation in predicted values occurs prominently near boundaries where the expected output values are highest. For instance, in Figure 4.4(c), which illustrates displacement in the x-direction (u_x), the left boundary is constrained in the x-direction but free in the y-direction. Traction applied on the right boundary induces movement in the positive x-direction, causing displacement (u_x) to increase from left to right, with maximum displacement at the bottom-right corner and higher errors along the bottom boundary. Similarly, displacement in the y-direction (u_y), where the left boundary is free to move, shows more pronounced movement and consequently higher errors.



(a) Predicted σ_{xx} (b) Reference σ_{xx} (c) Absolute error σ_{xx}

(d) Predicted σ_{yy} (e) Reference σ_{yy} (f) Absolute error σ_{yy}

(g) Predicted σ_{xy} (h) Reference σ_{xy} (i) Absolute error σ_{xy}

Figure 4.5: Plane stress problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of σ_{xx}, σ_{yy} and σ_{xy} respectively

The noticeable stress concentration near the hole in our quarter-plate scenario is primarily due to the presence of the hole. Solid mechanics principles dictate that holes, notches, or sharp corners induce high-stress concentrations, affecting displacement and strain fields. In our setup, the plate is constrained along the left and bottom boundaries in the x and y directions, respectively, while the arc boundary (hole) remains unconstrained and free. This difference in boundary conditions causes significant deformation around the arc. Traction applied in the positive x-direction causes the free arc boundary to move along the x-direction, leading to stress accumulation and concentration near the corner where the

left boundary meets the arc. The observed stress distribution near the corner of the left boundary and the arc highlights the complex interaction between applied loading, boundary conditions, and the capabilities of the PINN model to capture these phenomena. Figure 4.5(a) visually depicts the stress concentration phenomenon, while figure 4.5(c) illustrates the error associated with high-stress concentrations at this corner. A similar pattern is observed for stress in the y-direction (σ_{yy}), as shown in figure 4.5(d) and figure 4.5(f). This emphasizes the need for more precise training of the PINN model to achieve the desired accuracy in stress predictions. It is important to consider the accuracy of the stresses in the reference solution, as mesh refinement in Abaqus is crucial for obtaining precise results that can be meaningfully compared with PINN predictions. Further analysis addressing this comparison is presented in Section 4.1.3.

Evaluation of Results for Soft and Hard Constraints

In the section 2.2.3, the fundamental concepts of soft and hard constraints were introduced. In this study, we explore the impact of training the PINN model with displacement boundary conditions, implemented both as soft and hard constraints, comparing them against the reference solution.

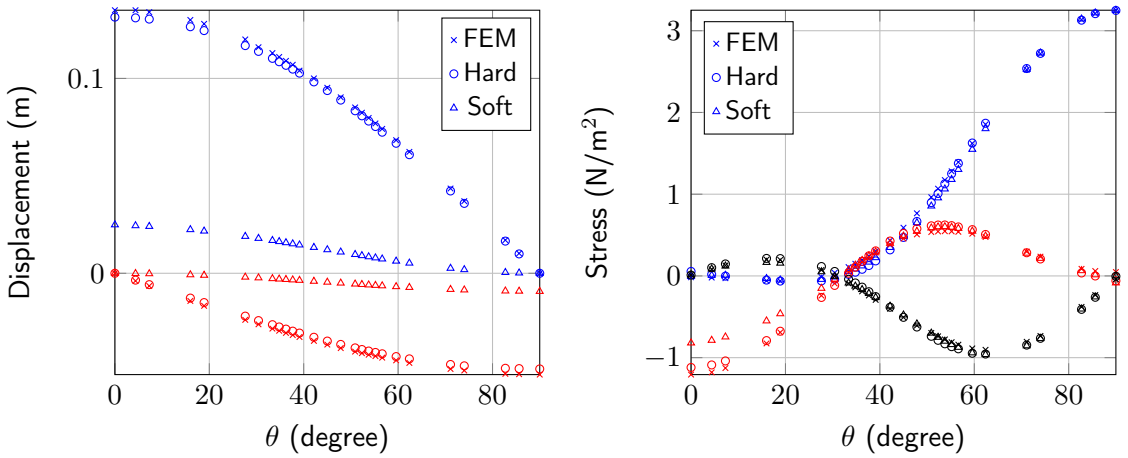
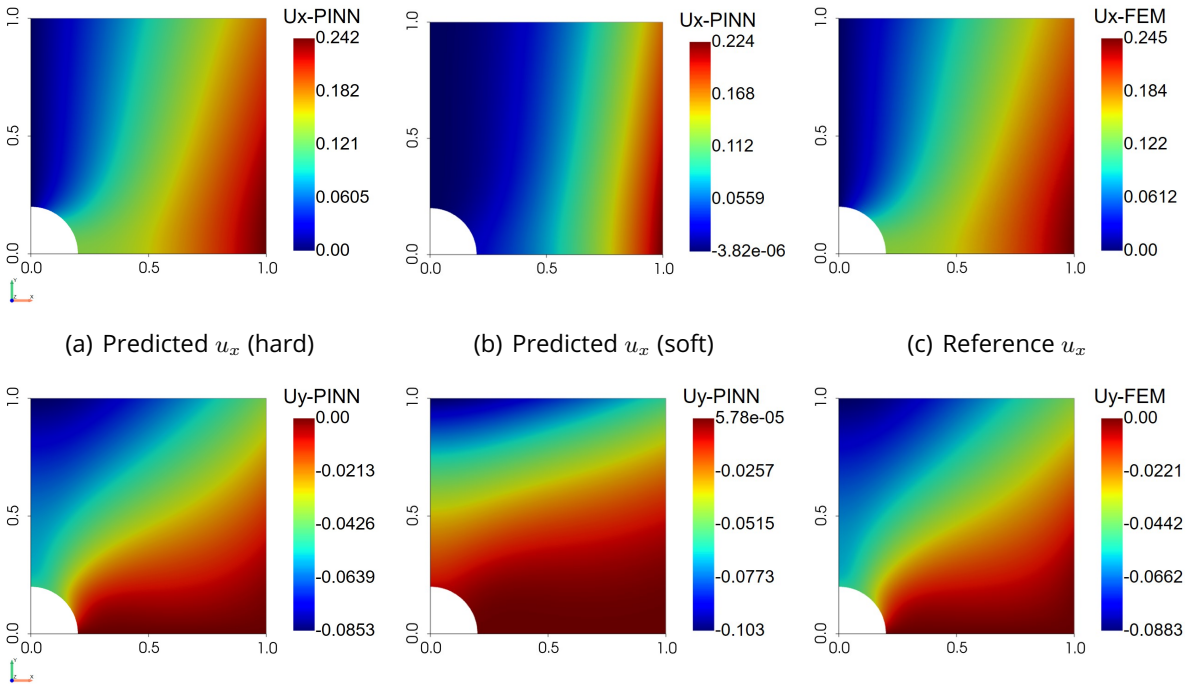


Figure 4.6: Soft and hard constraints. (a) The displacements u_x and u_y in x and y directions are represented with blue and red respectively, (b) The stresses σ_{xx}, σ_{yy} and σ_{xy} are represented with blue, red and black respectively.

The comparison plot depicted in figure 4.6(a) and 4.6(b) showcases the solutions obtained from FEM and PINNs, where displacement boundary conditions are enforced as soft and

hard constraints respectively. The output displacements and stresses values on the circular quarter arc are compared with the reference solution (FEM solution) and predicted outputs using hard and soft constraints respectively. Notably, employing displacement boundary conditions as hard constraints yields significantly better results compared to their implementation as soft constraints. Figure 4.7 illustrates the predicted solution for displacements obtained from the PINN model using hard and soft constraints, and reference solutions respectively. A significant difference between the results can be observed in figure 4.7(a) and figure 4.7(a). The PINN model with soft constraints successfully predicts the boundary pattern on the left, where the constraint enforces a zero value. However, within the domain, achieving accurate solutions becomes challenging with the current number of iterations. A similar observation applies to the displacement in the y-direction, u_y.



(a) Predicted u_x (hard) (b) Predicted u_x (soft) (c) Reference u_x

(d) Predicted u_y (hard) (e) Predicted u_y (soft) (f) Reference u_y

Figure 4.7: Plane stress problem with hard and soft constraints: (a) and (d) are the predicted solutions using hard constraints; (b) and (e) are the predicted solutions using soft constraints; (c) and (f) are the reference solutions, for the value of u_x and u_y respectively

Hard constraints rigorously ensure the satisfaction of boundary conditions throughout the domain, as specified in Equation (2.19). The soft constraints solely guarantee boundary conditions only along the specified boundary, disregarding values in other regions. Consequently, training under soft constraints may require more iteration and precise training to

achieve the desired accuracy or may even fail to converge, depending upon problem complexity. The effect on stress prediction is minimal as shown in figure 4.7(b), since stresses are currently enforced as soft constraints in both scenarios. Implementing stresses as hard constraints would involve a more complex approximation function, requiring further investigation. In research, Dirichlet boundary conditions for displacement are commonly implemented as hard constraints, while stresses are often treated as soft constraints. This study has not extensively explored the analytical formulation and implementation of stresses as hard constraints due to the thesis scope.

Analysis on Hardware Efficiency

The hardware structure and version utilized for this study are outlined in table 4.2. The initial study utilized consumer-grade hardware, specifically a laptop CPU and GPU. This hardware configuration was suitable for training the PINN model for two-dimensional and relatively simple models. However, as the complexity of the models increased, particularly for complex two-dimensional or three-dimensional geometries, it became evident that the consumer-grade hardware was inadequate. These more complex simulations necessitated higher computational power and memory resources. Therefore, all the three-dimensional simulations and complex two-dimensional simulations were performed on a High-Performance Computing (HPC) GPU Cluster at the NHR Center [6] of TU Dresden, which provided the necessary computational capabilities to handle the increased complexity and size of the simulations.

Table 4.2 outlines the computation time required on different hardware for the same neural network parameters, training points, and iterations. Training neural networks on a CPU is generally less efficient compared to using a GPU [47]. The computation time for training on an HPC cluster is almost 3 times faster compared to consumer hardware GPU. However, for more complex problems and 3D problems, this difference increases as the memory requirement increases and the consumer hardware becomes inefficient for 3D problems.

Table 4.2: Overview of Hardware specifications

Hardware	Details	Training time (s)
Laptop - CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz, RAM: 8GB	17590.41
Laptop - GPU	NVIDIA GeForce GTX 1660 Ti, Memory: 6GB	4144.27
HPC GPU Cluster	NVIDIA A100-SXM4, Memory: 40GB	1357.23

4.1.3 Cantilever Beam

In this subsection, we will evaluate the results for a cantilever beam subjected to uniform traction force, $\bar{t}$ = 1 N/m^2 on the top surface. The geometry and problem definition of the 2D cantilever beam is shown in figure 4.8. The beam has an arbitrary length, L = 1 m, and height, H = 0.2 m. The predicted solutions from the PINN model are evaluated and further a comparison with the reference solution from Abaqus is performed for different mesh sizes.

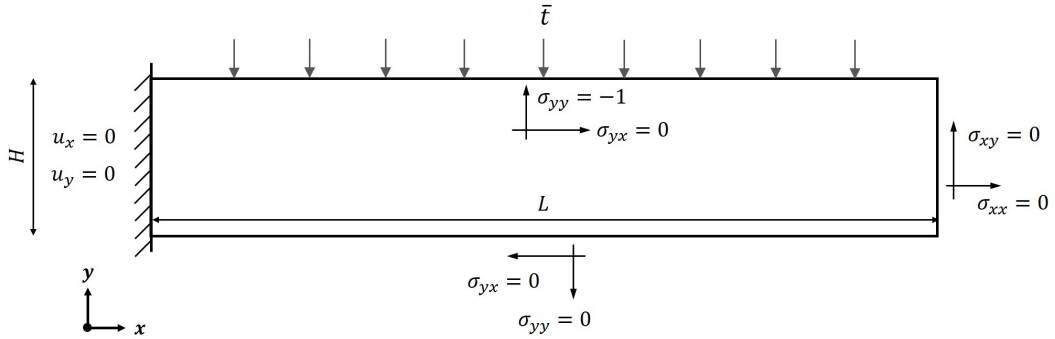
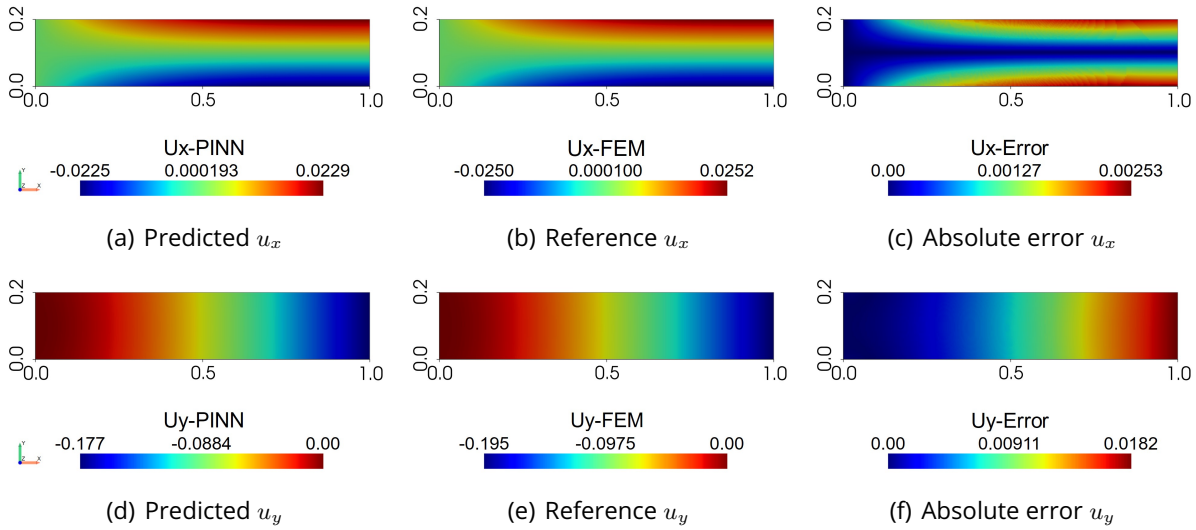


Figure 4.8: Displacement boundary conditions and traction conditions on a cantilever beam. The material parameters are: E = 1000 Pa, $\nu = 0.3$

The PINN model was trained using 7000 points, comprising 6000 points within the domain and 1000 points on the boundary. Due to the complexity of the output solutions and boundary conditions, the model exhibited slower convergence and required a large number of iterations for training to achieve the desired accuracy. Specifically, the PINN model was trained for 1.5 million iterations, utilizing 300,000 iterations with the ADAM optimizer and 1.2 million iterations with LBFGS. This problem highlighted that the complexity of the output solutions and boundary conditions can result in slower convergence and reduced performance. This aspect will be further investigated and discussed in the subsequent section of this study.

The results are shown in figure 4.9 and 4.10, depicting the predicted solution alongside reference solutions and the absolute error between them for various outputs. In this problem, the beam is fixed at one end, constrained in both the x and y directions, while the other end is free. As a result, the displacement distribution is highest at the free end and decreases towards the fixed end, as illustrated in figure 4.9(b) and 4.9(e). When the beam bends under the applied traction force, internal stresses develop within the material to resist bending and maintain equilibrium. This leads to high-concentration stresses near the

fixed boundary. The stress distribution along the cantilever beam varies, with maximum stress typically occurring at or near the fixed end. Conversely, the free end experiences either minimal stress or stress-free conditions. The distribution of normal stresses in the x and y directions, as well as shear stress, is depicted in figure 4.10(b), 4.10(e), and 4.10(h), respectively.

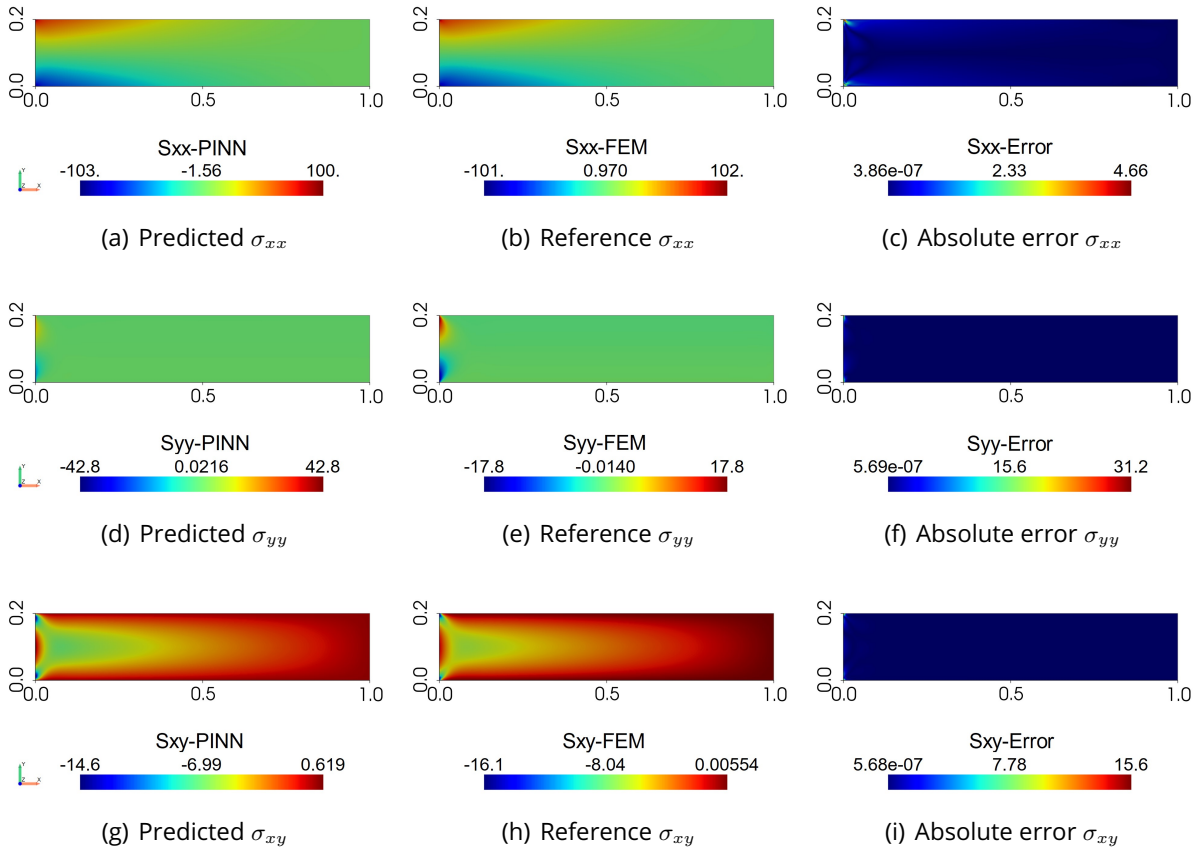


(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

Figure 4.9: Cantilever beam problem: (a), (d) are the predicted solutions; (b), (e) are the reference solutions; (c), (f) are the absolute differences between the reference and predicted solutions, for the value of u_x and u_y respectively

Upon reviewing all the predicted outputs, it is evident that certain results exhibit high accuracy, while others show minimal or significant deviations. For instance, the stress components σ_{xx} and σ_{xy} demonstrate good accuracy, whereas the displacement components u_x, u_y, and the stress σ_{yy} exhibit significant deviations. The PINN model predicted the distributions of displacement and stresses but did not converge to the reference solution. This is evident from figure 4.9(c) and 4.9(f), where significant deviations in predicted displacements are observed near the top and bottom boundaries for u_x, and near the right boundary for u_y where displacements are maximum (absolute value). The corresponding errors are also depicted in these figures. To further evaluate the accuracy of the predicted outputs, additional assessments were conducted by comparing them with reference solutions obtained from FEM simulations in Abaqus, employing various mesh sizes. The comparison in table 4.3 presents the predicted solution from the PINN model against reference solutions from Abaqus simulations with varying mesh densities: coarse mesh (6771 nodes), fine mesh (12,801 nodes), and very fine mesh (201,201 nodes). The table shows the pre-



(a) Predicted σ_{xx} (b) Reference σ_{xx} (c) Absolute error σ_{xx}

(d) Predicted σ_{yy} (e) Reference σ_{yy} (f) Absolute error σ_{yy}

(g) Predicted σ_{xy} (h) Reference σ_{xy} (i) Absolute error σ_{xy}

Figure 4.10: Cantilever beam problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of σ_{xx}, σ_{yy} and σ_{xy} respectively

dicted outputs' maximum (left) and minimum (right) values. The analysis indicated that the predicted solutions from the PINN model outperformed the reference solution with a coarse mesh and were somewhat close to the solution with a fine mesh, but still noticeably deviated from the very fine mesh reference solution.

Notably, while the predicted stresses aligned somewhat with the fine mesh reference solution, the predicted displacements exhibited larger deviations. This discrepancy suggests that further convergence of the PINN model, resulting in corrections to displacements (e.g., from 0.0223 to 0.0252 in u_x and from -0.179 to -0.195 in u_y), could lead to improved accuracy in predicted stresses comparable to the very fine mesh reference solution. Another noteworthy observation is the significant discrepancy in the predicted output of σ_{yy}, which remains substantially higher than the reference solution even with a very fine mesh. This discrepancy suggests that further mesh refinement might reveal additional insights. The

variation in accuracy across different predicted outputs indicates that some are learned more effectively than others. This discrepancy is likely influenced by the distribution and non-linearity of the outputs, along with factors such as weight initialization and the distribution of weight losses. Additionally, the imbalance in losses can impact the model's performance, potentially affecting convergence and accuracy. This underscores the importance of optimizing the weight initialization and loss distribution to enhance the training and overall performance of the model.

Table 4.3: Comparison between PINN model and Abaqus model.

Parameter	PINN model	Abaqus model		
	predicted solution (7000 training points)	6771 nodes	12,801 nodes	201,201 nodes
u_x	0.02247, -0.02285	0.02526, -0.02497	0.02525, -0.02496	0.02524, -0.02495
u_y	0, -0.1768	0, -0.1954	0, -0.1953	0, -0.1952
σ_{xx}	100.06, -103.17	95.57, -94.00	102.45, -100.51	140.34, -136.39
σ_{yy}	42.81, -42.76	16.35, -16.43	17.81, -17.84	24.76, -24.46
σ_{xy}	0.6193, -14.59	0.00386, -14.98	0.00554, -16.08	0.00645, -26.70

4.1.4 Effect of Boundary Condition on Annular Quarter Disk

In the previous section, it was noted that the PINN model required a longer training duration due to the complexity of the problem. The non-linearity of the output solution heavily influences the convergence and performance of the PINN model. In this section, we will train an annular quarter disk geometry under two scenarios: one with roller boundary conditions on the left and bottom boundaries, and another with fixed boundary conditions on both the left and bottom boundaries. In both cases, the inner boundary experiences internal pressure implemented as traction force ($\bar{t}$), while the outer boundary remains traction-free. Both PINN models are trained with identical material parameters and neural network hyperparameters, incorporating suitable displacement and traction boundary conditions as depicted in figure 4.11. The arbitrary dimensions of the geometry are; the inner radius of the quarter disk is 0.5 m and the outer radius is 1 m. This study explores how the PINN model behaves under different boundary conditions and complexities of output solutions for similar geometries.

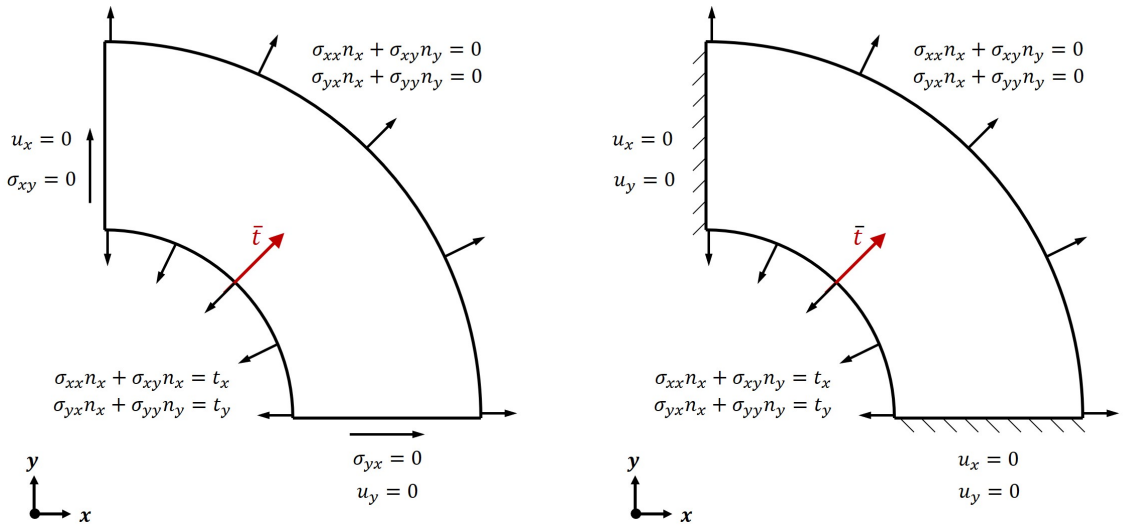
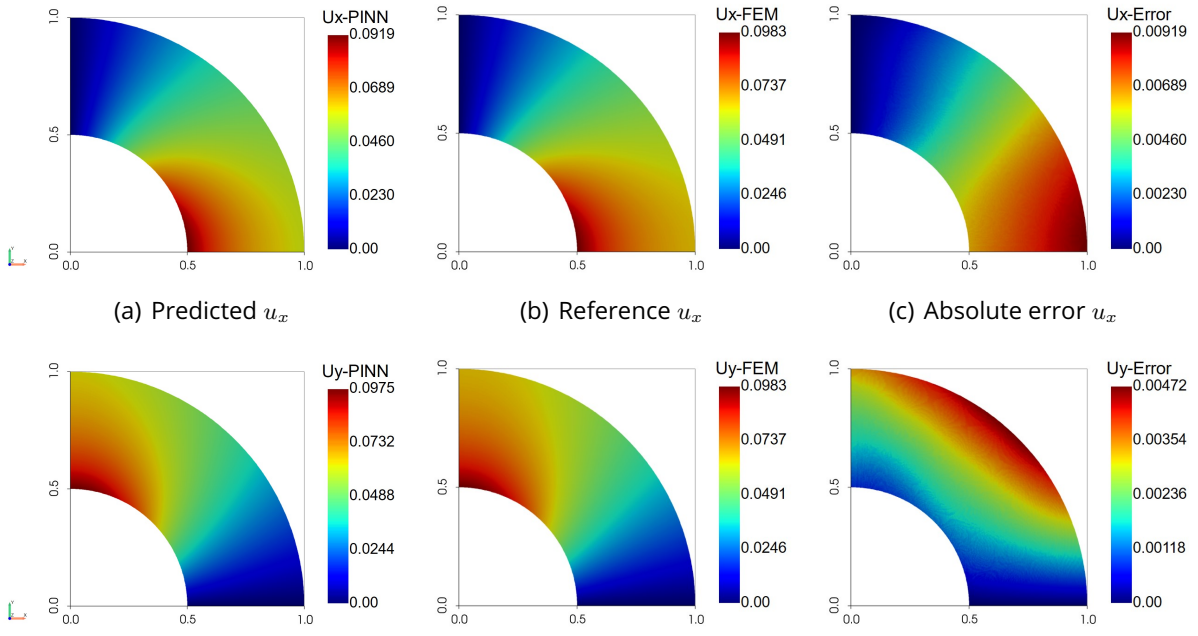


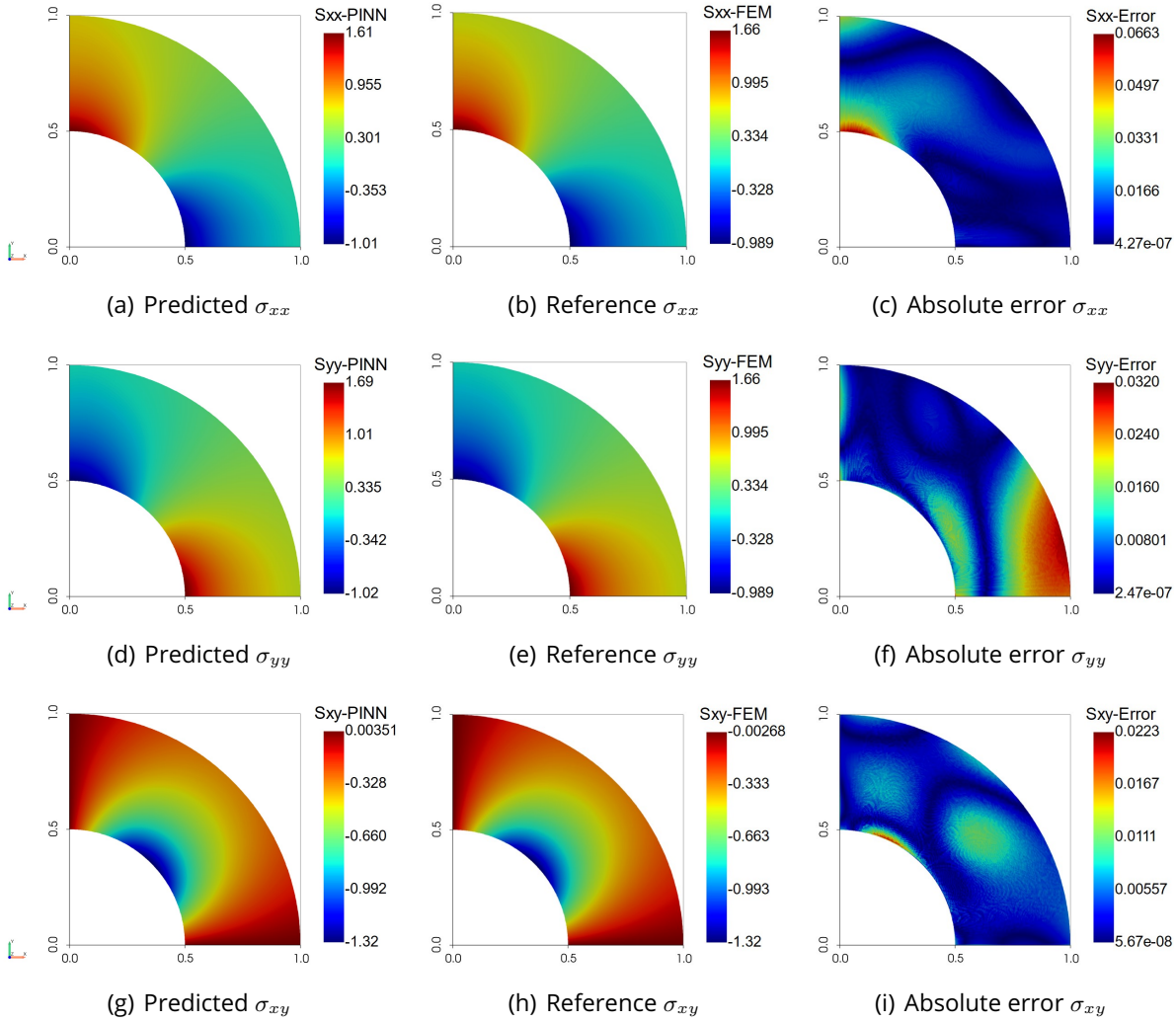
Figure 4.11: The configuration of different boundary conditions, (a) Roller boundary condition, (b) Fixed boundary condition. The material parameter are: E = 10 Pa, $\nu = 0.3$



(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

Figure 4.12: Roller boundary condition problem: (a), (d) are the predicted solutions; (b), (e) are the reference solutions; (c), (f) are the absolute differences between the reference and predicted solutions, for the value of u_x and u_y respectively.



(a) Predicted σ_{xx} (b) Reference σ_{xx} (c) Absolute error σ_{xx}

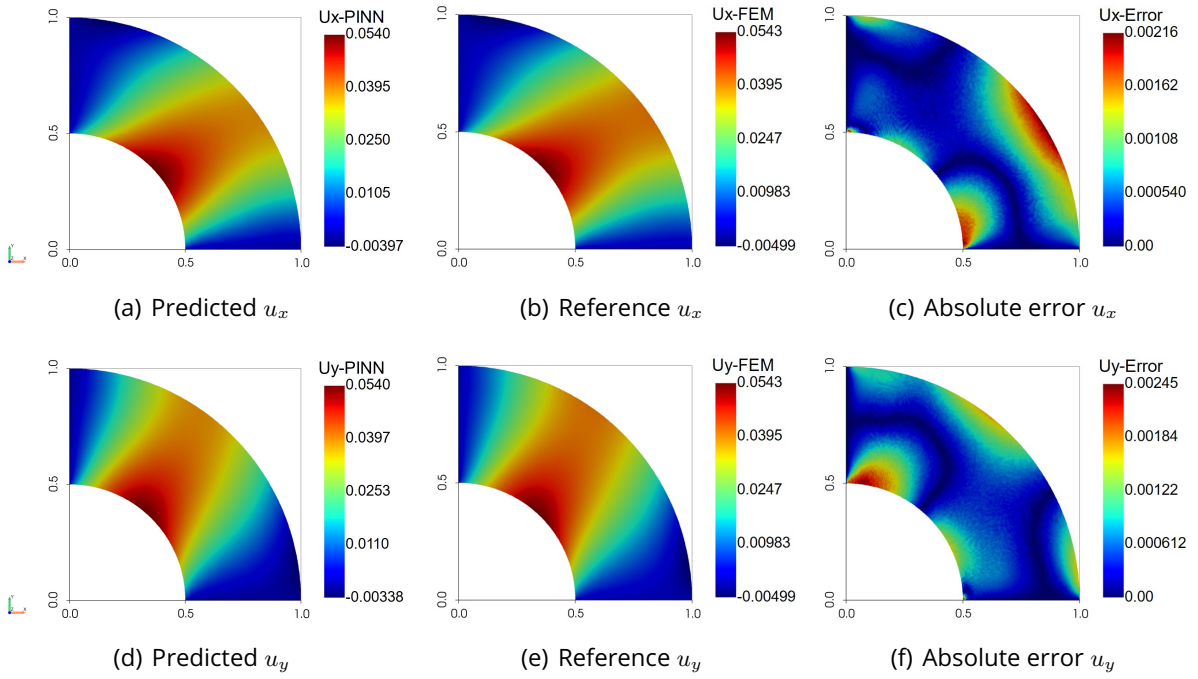
(d) Predicted σ_{yy} (e) Reference σ_{yy} (f) Absolute error σ_{yy}

(g) Predicted σ_{xy} (h) Reference σ_{xy} (i) Absolute error σ_{xy}

Figure 4.13: Roller boundary condition problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of σ_{xx}, σ_{yy} and σ_{xy} respectively.

In figure 4.13 and 4.15 the predicted solution of stresses and the reference solution from Abaqus are displayed, accompanied by their absolute errors both for roller boundary conditions and fixed boundary conditions respectively. Significant differences in stress concentration are observed between the roller boundary condition and the fixed boundary condition. For example, the stress distribution in the x-direction σ_{xx} can be seen in figure 4.13(b) and 4.15(b) for roller and fixed conditions, respectively. Stresses are notably concentrated under fixed boundary conditions, exhibiting a highly nonlinear and complex distribution. Similar observations can be made for σ_{yy} and σ_{xy}, where stresses are concentrated at the junction of the fixed corner and the free inner boundary. This concentration

of stresses is influenced by the abrupt change in boundary conditions and leads to intricate stress patterns. As a consequence, the deviation between the predicted and reference solutions is notably high in these regions of high-stress concentration. This can be observed in figure 4.15(c), 4.15(f), and 4.15(i), which display the absolute error for σ_{xx}, σ_{yy}, and σ_{xy} respectively under fixed boundary conditions. In contrast, the deviation in stress for roller boundary conditions is minimal.



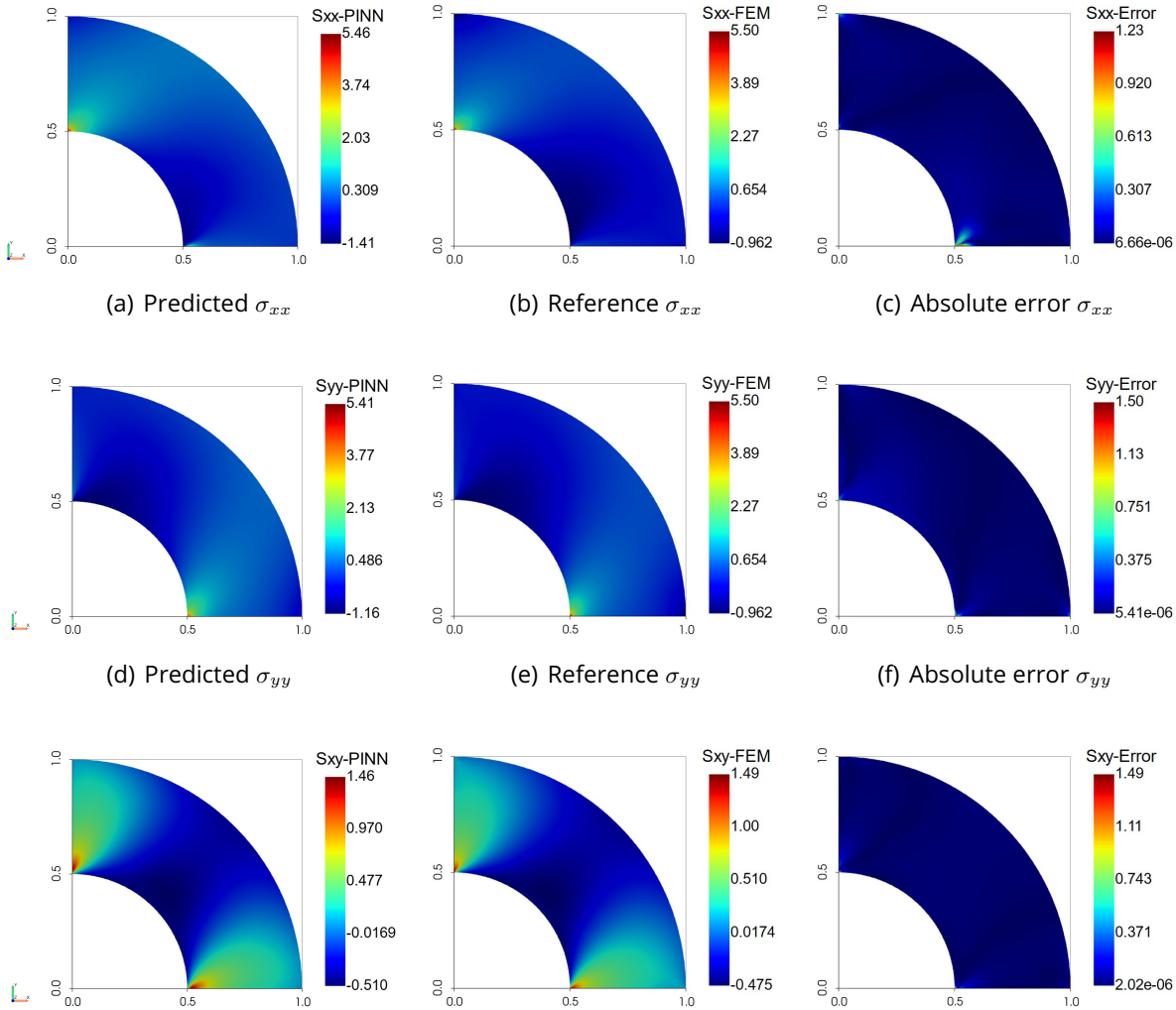
(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

Figure 4.14: Fixed boundary condition problem: (a), (d) are the predicted solutions; (b), (e) are the reference solutions; (c), (f) are the absolute differences between the reference and predicted solutions, for the value of u_x and u_y respectively

The presence of highly concentrated and nonlinear stress distributions under fixed boundary conditions underscores the complexity of accurately modeling and predicting these behaviors with the PINN model. The PINN model with roller boundary conditions achieved convergence in just 30,000 iterations, showcasing rapid convergence. Whereas, the model with fixed boundary conditions exhibited slower convergence and required a significantly larger number of iterations. The convergence of the PINN model with fixed boundary conditions took 1.3 million iterations and a significant amount of time to complete the training process. This convergence rate disparity highlights the influence boundary conditions on the optimization process. The observed difference in convergence rates can be attributed to the stricter constraints imposed by fixed boundary conditions, which inherently con-

strain the PINN model more rigidly. Consequently, the optimization process becomes more challenging, necessitating a larger number of iterations to satisfy these constraints accurately. Although the output ranges of the two models may vary, the fundamental observation remains consistent: fixed boundary conditions lead to slower convergence in PINN models due to their more rigorous constraints. This finding underscores the importance of considering boundary conditions in model optimization and highlights the need for tailored optimization strategies to address convergence challenges.



(a) Predicted σ_{xx} (b) Reference σ_{xx} (c) Absolute error σ_{xx}

(d) Predicted σ_{yy} (e) Reference σ_{yy} (f) Absolute error σ_{yy}

(g) Predicted σ_{xy} (h) Reference σ_{xy} (i) Absolute error σ_{xy}

Figure 4.15: Fixed boundary condition problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of σ_{xx}, σ_{yy} and σ_{xy} respectively

4.2 Evaluation and Validation of Three Dimensional Geometries

In this section, we will delve into the outcomes of three-dimensional geometries. It will cover the accuracy of the results, the impact of intricate output solutions, and the challenges encountered in achieving convergence, particularly in cases involving substantial parameters and fixed boundary conditions. To evaluate the stresses, the analysis employs the von Mises stress criterion, which is expressed as,

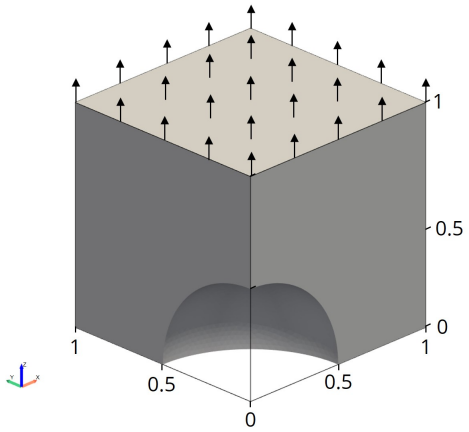
$$\sigma_{\mathrm{v}M} = \sqrt{\frac{1}{2}\left[(\sigma_{xx}-\sigma_{yy})^2+(\sigma_{yy}-\sigma_{zz})^2+(\sigma_{zz}-\sigma_{xx})^2\right]+3\left(\sigma_{xy}^2+\sigma_{yz}^2+\sigma_{zx}^2\right)} \tag{4.4}$$

This criterion provides a measure of the equivalent stress experienced by a material undergoing complex loading conditions, accounting for both normal and shear stresses across different directions.

4.2.1 Cube with a Spherical Hole

In this subsection, a cubic structure with a spherical void undergoes uniform traction force, t = 1 N/m^2 along the z-direction on the top surface, as depicted in figure 4.16(a). The cube is of unit dimension and the spherical void has a radius of 0.5 m. The roller boundary conditions are employed at the surfaces where x = 0, y = 0, and z = 0. The traction condition is applied at the top surface where z = 1, and all the other surfaces are traction-free. These boundary conditions are outlined in figure 4.16(b).

Solving the three-dimensional linear elasticity problem with a PINN model demands extensive iterations and computational resources due to numerous network hyperparameters, and the complexity of calculating three-dimensional PDEs and their gradients. Additionally, the model faces challenges stemming from disparate output solutions, requiring many loss functions to handle PDEs, boundary conditions, and variations in output scales. These factors often hinder the training process, leading to slow convergence or even failure to reach a solution. The PINN model was trained using approximately 8800 training points, employing a neural network architecture comprising 9 MLPs for each output. Each MLP consists of 30 neurons spread across 6 layers. The model was trained for 550,000 iterations using the ADAM optimizer to achieve the obtained accuracy, which required approximately 21 hours of training time.



(a)

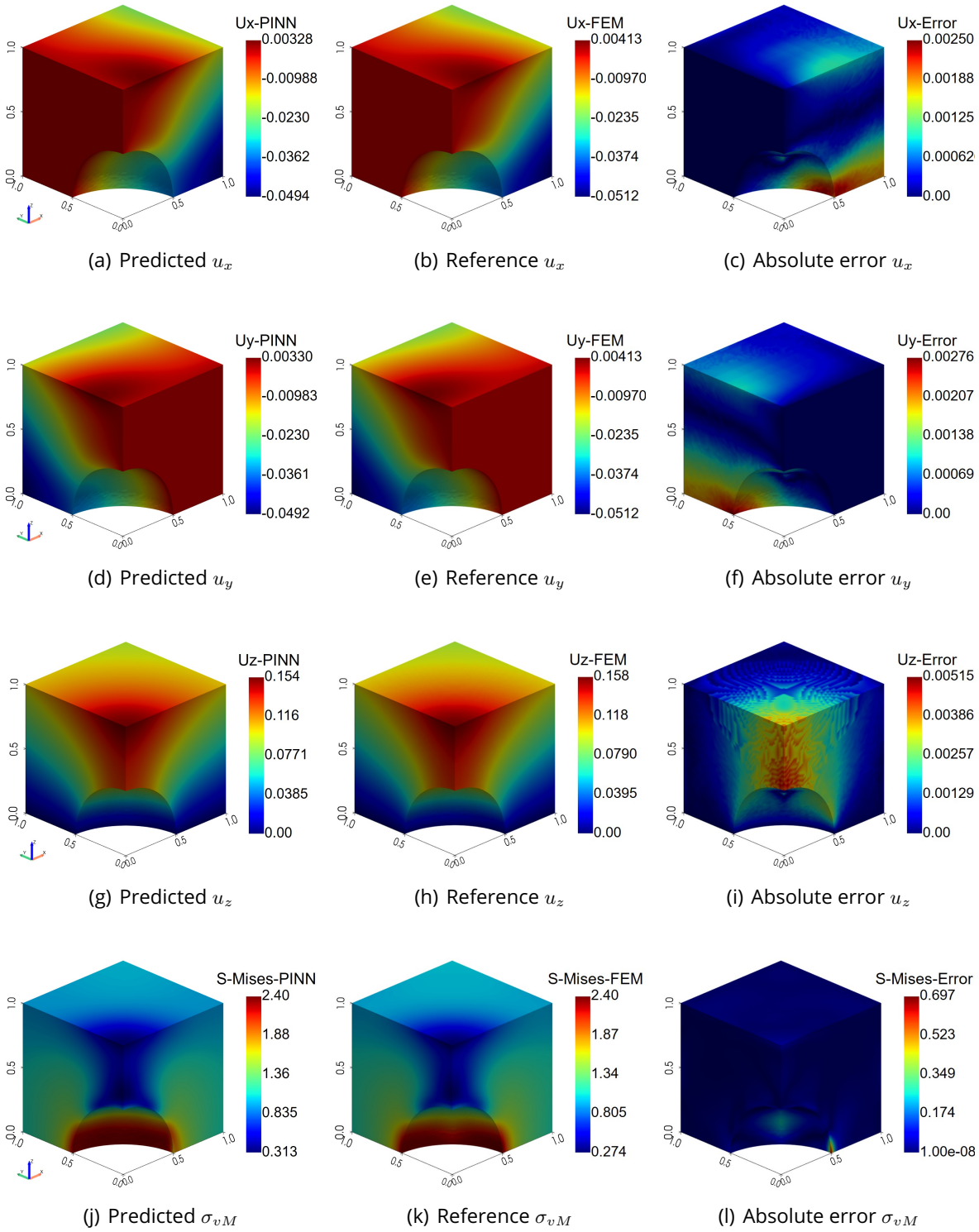
Boundary	Condition
x = 0	$u_x = 0,\ \sigma_{xy} = 0,\ \sigma_{xz} = 0$
y = 0	$u_y = 0,\ \sigma_{yx} = 0,\ \sigma_{yz} = 0$
z = 0	$u_z = 0,\ \sigma_{zx} = 0,\ \sigma_{zy} = 0$
x = 1	$\sigma_{xx} = 0,\ \sigma_{xy} = 0,\ \sigma_{xz} = 0$
y = 1	$\sigma_{yy} = 0,\ \sigma_{yz} = 0,\ \sigma_{yz} = 0$
z = 1	$\sigma_{xy} = 1,\ \sigma_{zx} = 0,\ \sigma_{zy} = 0$
spherical arc	$\sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z = 0,$ $\sigma_{yx}n_x + \sigma_{yy}n_y + \sigma_{yz}n_z = 0,$ $\sigma_{zx}n_x + \sigma_{zy}n_y + \sigma_{zz}n_z = 0,$

(b)

Figure 4.16: (a) Cube subjected to uniform traction force, (b) Displacement and traction conditions on different boundaries. The material parameters are: E = 10 Pa, $\nu = 0.3$

The predicted solution from the PINN model alongside the reference solution and the absolute error between them is depicted in figure 4.17. The figure demonstrates that the PINN model could predict the output solution with minimal error in predicting displacements and stresses.

We will first consider the vertical displacement, u_z to delve into the analysis. In our setup, the bottom surface at z = 0 is fixed, while the top surface at z = 1 is free. A traction force is applied in the positive z-direction, acting as a pulling force. Consequently, the maximum displacement is expected to occur at the top surface. Furthermore, the spherical boundary surface is unrestricted, allowing for maximum displacement near the corner where the top of the spherical surface remains free. This displacement pattern is evident in figure 4.17(g). Similarly, in the plot of u_x and u_y, the displacement is more pronounced at the free end (where no constraints are applied) and approaches zero on the restricted surfaces as depicted in figure 4.17(a) and 4.17(d) respectively. The displacement distribution observed aligns with the expected behavior based on the applied loading and boundary constraints. The absolute error is higher at the corners where the displacements are more pronounced, as depicted in figure 4.17(c), 4.17(f), and 4.17(i). The mean absolute error for displacements in x, y and z direction are as follows: e_{u_x} = 3.88×10^{-4}, e_{u_y} = 4.38×10^{-4} and e_{u_z} = 7.62×10^{-4} respectively.



(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

(g) Predicted u_z (h) Reference u_z (i) Absolute error u_z

(j) Predicted σ_{vM} (k) Reference σ_{vM} (l) Absolute error σ_{vM}

Figure 4.17: 3D cube problem: (a), (d), (g), (j) are the predicted solutions; (b), (e), (h), (k) are the reference solutions; (c), (f), (i), (l) are the absolute differences between the reference and predicted solutions, for value of u_x, u_y, u_z, and σ_{vM} respectively

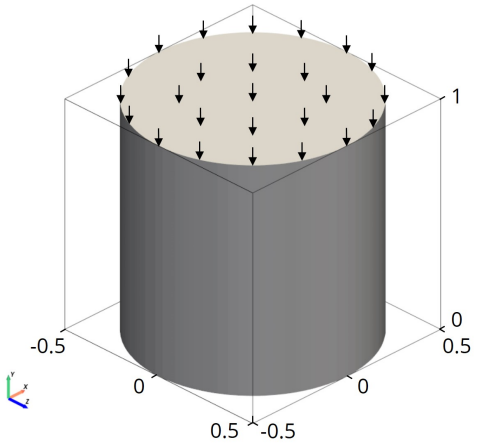
For stress analysis, von Mises stress was calculated instead of evaluating normal and shear stresses in each direction. The von Mises stress is computed using the normal and shear stresses (σ_{xx}, σ_{yy}, σ_{zz}, σ_{xy}, σ_{yz}, σ_{zx}) according to Equation (4.4). Figure 4.17(j) illustrates that the stress is highest at the bottom surface, which is constrained. Both the predicted and reference von Mises stress values exhibit similarity with a mean absolute error of $e_{\sigma_{vM}} = 9.04 \times 10^{-3}$, although there is a noticeable deviation in one of the corners of the cube near the bottom, as seen in the absolute error plot in figure 4.17(l). This discrepancy could potentially be attributed to various factors, such as the inadequacy of training points in specific regions or inaccuracies in the estimation of surface normals. In stress computations, accurate surface normals are crucial for calculating the stress. Errors or inaccuracies in estimating surface normals can lead to misalignment of stress components, impacting the overall accuracy of stress predictions. Further analysis is needed to precisely identify the underlying cause and address any shortcomings in the model's performance.

4.2.2 Convergence Difficulties for Complex Model: 3D Cylinder and 3D Cantilever Beam

In this subsection, we explore the impact of fixed boundary conditions on the convergence and accuracy of solution outcomes across two distinct geometries. Building upon the findings and validations presented in Section 4.1.4, it is evident that models constrained by fixed boundary conditions pose significant challenges due to inherent complexity and nonlinearity in the distribution of output parameters, particularly stresses in the context of linear elasticity analysis. The first geometry under consideration is a cylindrical structure fixed at its bottom surface and subjected to a constant traction force on its top surface, while the curved surface remains traction-free. The second geometry involves a three-dimensional cantilever beam fixed at one end and subjected to a body force. This setup introduces complexities associated with structural behavior under specific loading conditions, where the interaction between applied forces and fixed supports significantly influences the resulting stress and deformation profiles. The formulation, boundary conditions, and outcomes of both geometries are thoroughly discussed in this section, shedding light on the challenges posed by fixed boundary conditions and their implications for accurate model predictions in linear elasticity analysis. Through detailed analysis and interpretation of results, we aim to elucidate the behavior of these systems under different constraints and loading scenarios, offering insights into the underlying physics and numerical aspects of the problem.

3D Cylinder Subjected to Uniform Traction Force

In this analysis, we will delve into evaluating the 3D cylindrical geometry. The radius of the cylinder is 0.5 m and the height is 1 m. The cylinder is fixed at the bottom surface at y = 0, and a traction force of magnitude 1 N/m^2 is applied on the top surface at y = 1. The problem configuration and boundary conditions are visually depicted in Figures 4.18(a) and 4.18(b).



(a)

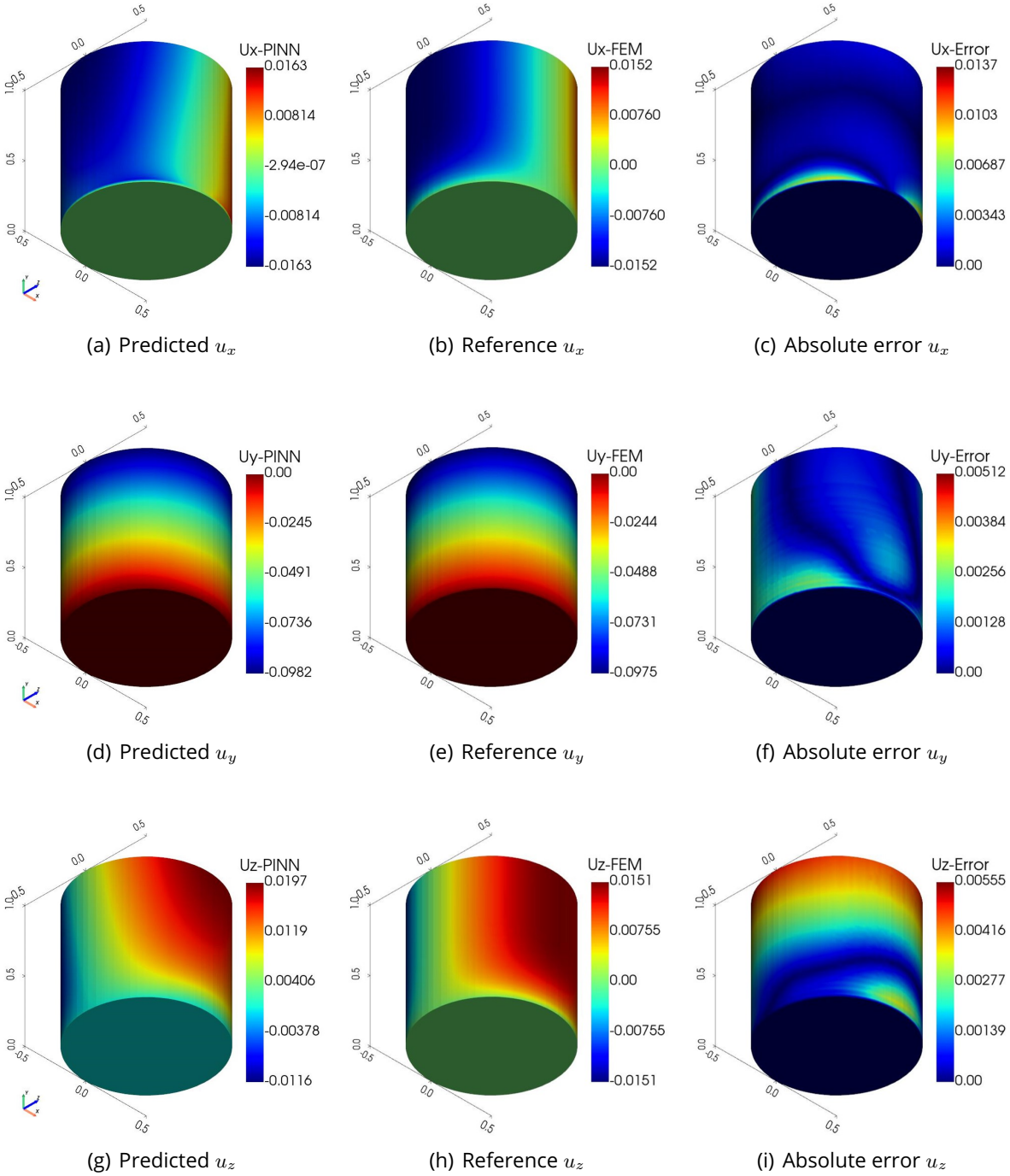
Boundary	Condition
y = 0	$u_x = 0$, $u_y = 0$, $u_z = 0$
y = 1	σ_{yy} = -1, $\sigma_{yx} = 0$, $\sigma_{yz} = 0$
cylindrical arc	$\sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z = 0$, $\sigma_{yx}n_x + \sigma_{yy}n_y + \sigma_{yz}n_z = 0$, $\sigma_{zx}n_x + \sigma_{zy}n_y + \sigma_{zz}n_z = 0$,

(b)

Figure 4.18: (a) 3D cylinder configuration, (b) Displacement and traction conditions on different boundaries. The material parameters are: E = 10 Pa, $\nu = 0.3$

Given the intricacy of this model, the PINN is structured with 160 neurons and 6 layers in each MLP, employing a total of 9 MLPs, each dedicated to an individual output parameter. Training models for three-dimensional geometries are computationally demanding, requiring extensive iterations and computational time. In this case, we employed 25,000 training points inside the domain and 2000 training points on the boundary, and the model was trained for 2.2 million iterations, which consumed approximately 257 hours ($\approx$ 10 days) to complete.

Despite the substantial computational investment, the predicted solutions do not completely align as expected with the reference solutions. The outcomes of the analysis are visualized in figure 4.19 and 4.20. The displacements are predicted more accurately compared to the stresses. The figures 4.19(a) and 4.19(d) demonstrate that the PINN model predicts the displacements u_x and u_y with minimal deviation. The notable errors are observed near the fixed boundary, as depicted in Figures 4.19(c) and 4.19(c).

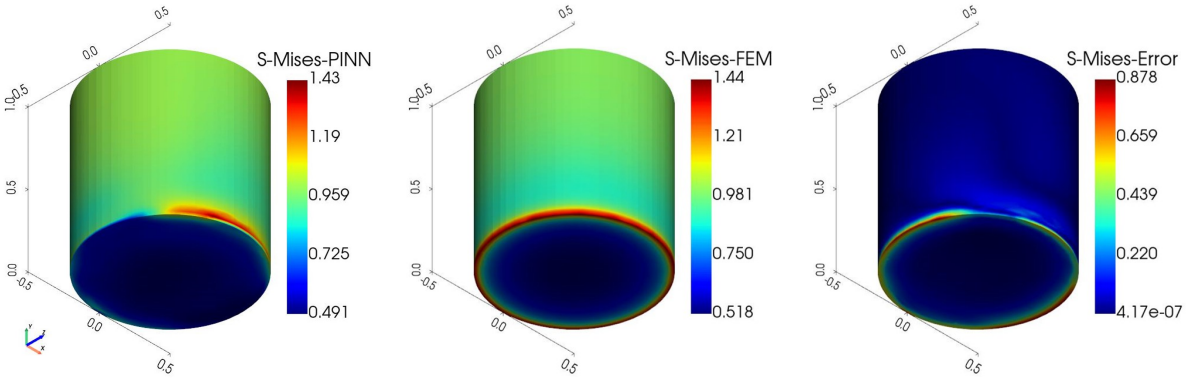


(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

(g) Predicted u_z (h) Reference u_z (i) Absolute error u_z

Figure 4.19: 3D Cylinder problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of u_x, u_y and u_z respectively



(a) Predicted σ_{vM} (b) Reference σ_{vM} (c) Absolute error σ_{vM}

Figure 4.20: 3D Cylinder problem: (a) is the predicted solution; (b) is the reference solution; (c) is the absolute difference between the reference and predicted solution, for the value of σ_{vM}.

In the case of u_z, significant deviation is observed, particularly near the top boundary where the positive displacement predicted by the model is significantly larger than the reference, while the predicted negative displacement is lower compared to the reference. The absolute error can be seen in figure 4.19(i). This error could likely be due to the limited number of training points. Furthermore, during the training process, the total training loss was on the order of 10^{-4}, but the total test loss was on the order of 10^{-1}. Upon monitoring the training process, it was observed that the test loss was high only for u_z, indicating that the model was overfitting. Overfitting could be attributed to various factors, such as insufficient training points on the boundary. Increasing the number of training points may lead to better results and mitigate overfitting issues.

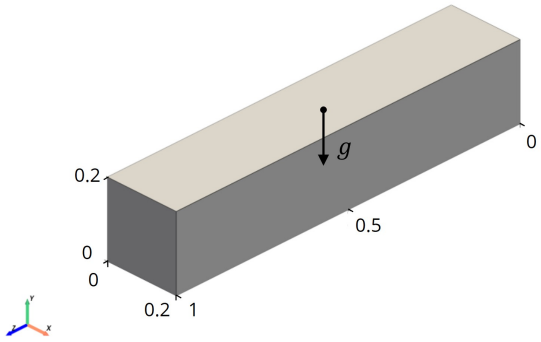
Moving to the analysis of stresses, let's consider the von Mises stress, which accounts for the complexity of stress distributions in different directions, including normal and shear stresses. The PINN model predicted the output stresses accordingly, yielding reasonably good results for some components like σ_{xx} and σ_{zz}, but less satisfactory results for σ_{yy} and other shear stresses. When von Mises stresses are calculated, these differences become apparent. The significant difference between the predicted and reference von Mises stresses can be observed in figure 4.20(a) and 4.20(b), particularly near the fixed boundary. Additionally, the stress distribution appears complex, with concentrations primarily near the fixed surface and along the circumference of the surface.

It is apparent from the predicted solutions that significant deviations exist, especially in stress distributions along the fixed boundary. However, the predictions for displacements

exhibit relatively better agreement with the reference solutions. The complexity of the model presents challenges for the PINN approach, especially in accurately predicting solutions at constrained boundaries. This underscores the necessity for further exploration into denser and larger neural network architectures, optimized training methodologies, and innovative approaches to reduce computational overhead.

3D Cantilever Beam Subjected to Body Force

In this analysis, we will discuss the three-dimensional cantilever beam subjected to a body force (e.g., gravity) and fixed at one end. The beam's dimensions, displacement, and traction boundary conditions are illustrated in figure 4.21(a) and 4.21(b) respectively. The beam has an arbitrary length of 1 m and a width and height of 0.2 m, it is fixed and z = 0, and the body force, g = 1 N/m^3 is applied in negative y-direction through the governing equation (2.23). The PINN model utilized a configuration with 160 neurons, 6 layers, and 9 MLPs. It was trained for 1.8 million iterations, which required approximately 190.8 hours ($\approx$ 8 days) to complete.



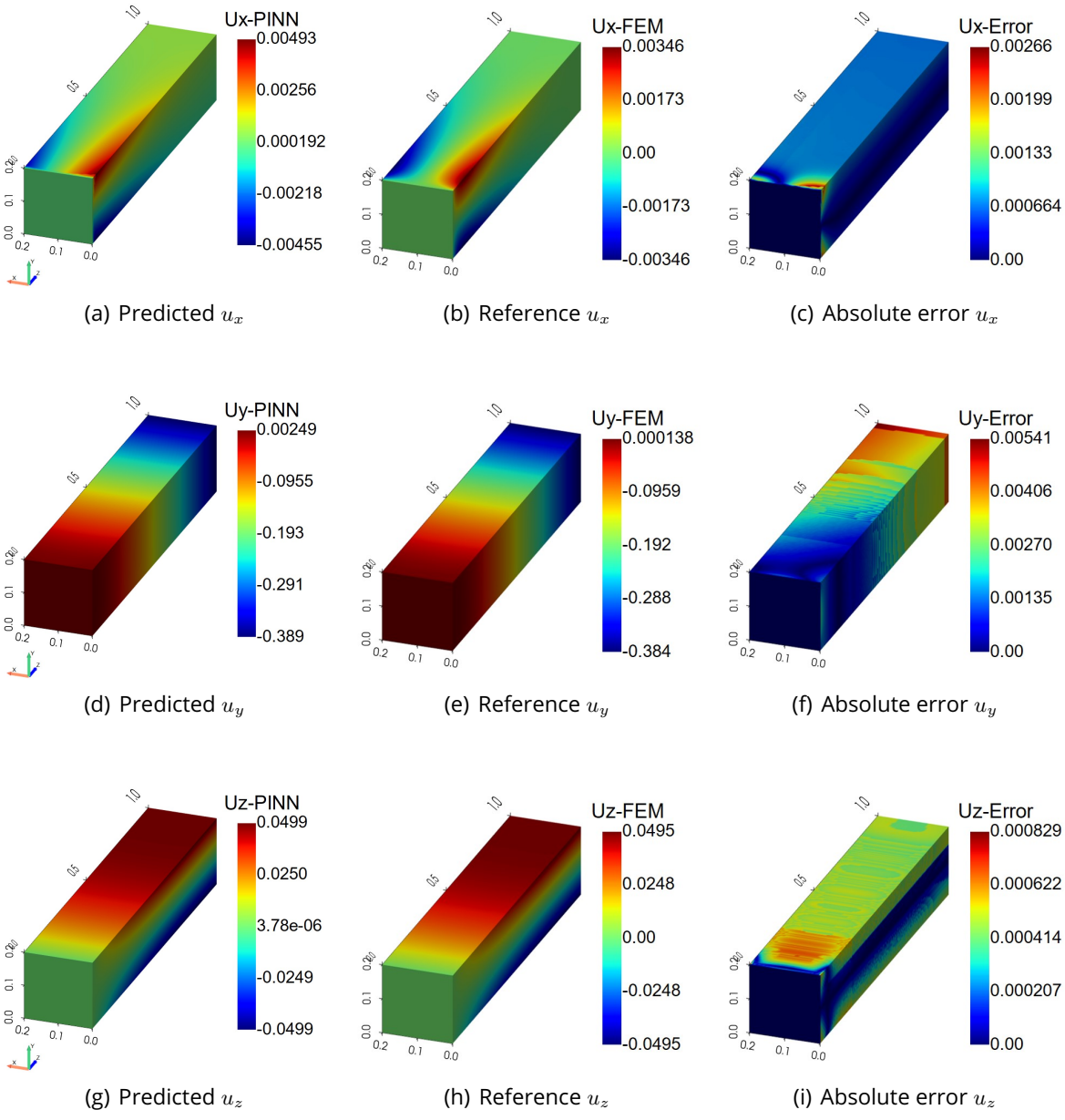
(a)

Boundary	Condition
x = 0, x = 0.2	$\sigma_{xx} = 0$, $\sigma_{xy} = 0$, $\sigma_{xz} = 0$
y = 0, y = 0.2	$\sigma_{yy} = 0$, $\sigma_{yx} = 0$, $\sigma_{yz} = 0$
z = 0	$u_x = 0$, $u_y = 0$, $u_z = 0$
z = 1	$\sigma_{zz} = 0$, $\sigma_{zx} = 0$, $\sigma_{zy} = 0$,

(b)

Figure 4.21: (a) 3D cantilever beam configuration, (b) Displacement and traction conditions on different boundaries

Similar to the previous cylindrical geometry, this model also encountered challenges with convergence during training, leading to inconsistencies in the predicted outputs. While some parameters were predicted with reasonable accuracy, others exhibited significant deviations from the reference solutions, as depicted in figure 4.22, 4.23 and 4.24.



(a) Predicted u_x (b) Reference u_x (c) Absolute error u_x

(d) Predicted u_y (e) Reference u_y (f) Absolute error u_y

(g) Predicted u_z (h) Reference u_z (i) Absolute error u_z

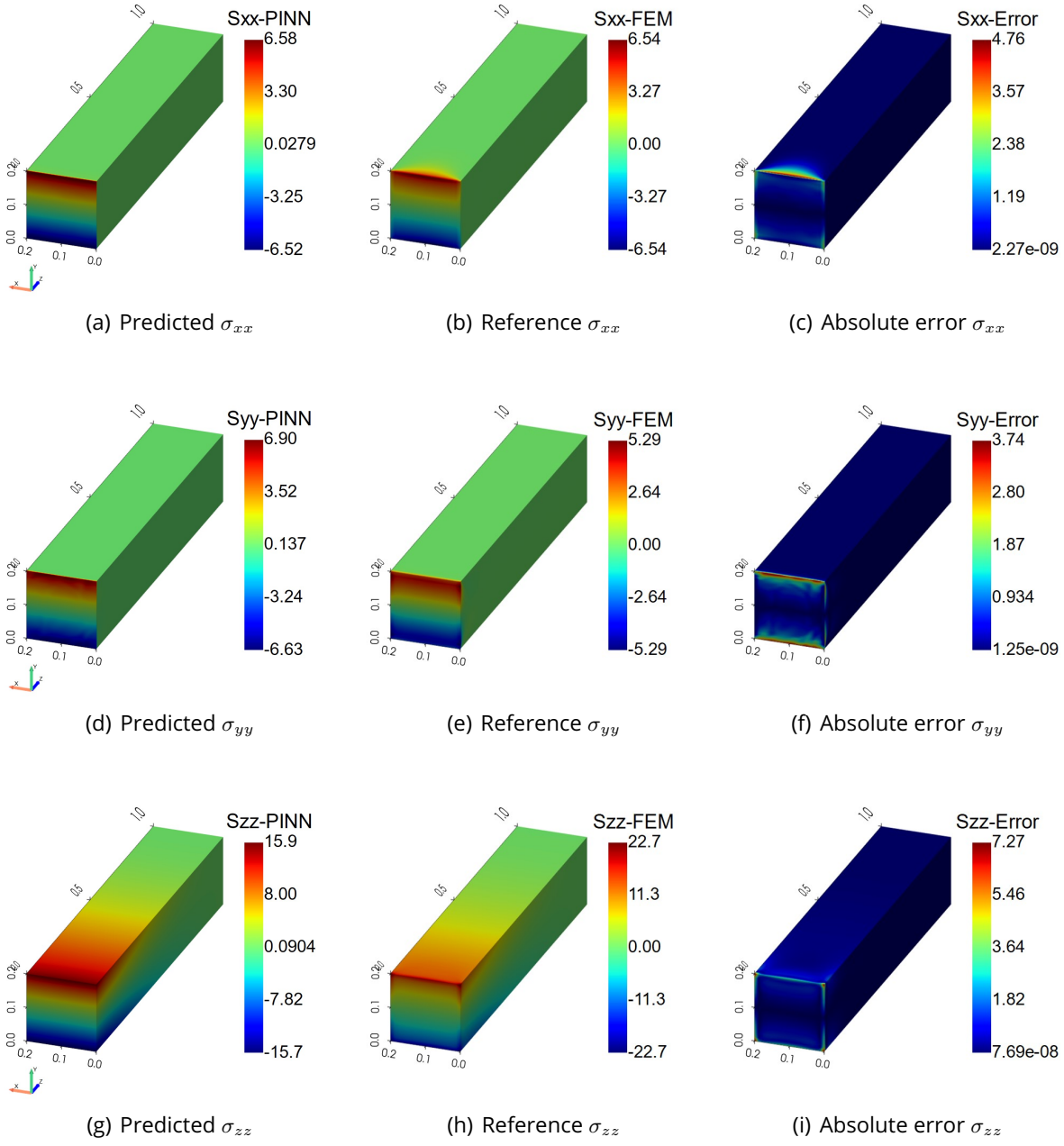
Figure 4.22: 3D Cantilever beam problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of u_x, u_y and u_z respectively

The cantilever beam problem is inherently complex, especially when extended to 3D scenarios. It is observed that the PINN model tended to overestimate displacements, particularly with significant deviations in the output value u_x compared to u_y and u_z which can be observed in figure 4.22(a), 4.22(d) and 4.22(g) respectively. The overestimation of u_x can be attributed to several factors. One potential reason could be the inherent complexities of modeling 3D structures, which introduce additional challenges compared to 2D analyses. The increased dimensionality of the problem can lead to more complex stress and strain distributions, affecting the model's predictive accuracy. Furthermore, the distribution of training points and the model architecture may not fully capture the intricate behaviors exhibited in 3D systems, contributing to the observed deviations.

For this analysis, we conduct a detailed evaluation of stress, including visual representations of both normal and shear stresses along with the von Mises stress. Due to fixed constraints, stress concentration is more pronounced on the fixed boundary. From figure 4.23 and 4.24, it can be observed that some stresses are accurately predicted by the PINN model, while others show significant deviations. For example, the stress σ_{xx} is accurately predicted, as depicted in figure 4.23(a), whereas substantial deviations exist in the normal stress σ_{zz}, indicating the need for further convergence, as shown in figure 4.23(i). Additionally, the stress σ_{yy} is overestimated, as shown in figure 4.23(d), with the corresponding absolute error illustrated in figure 4.23(f). The shear stresses are predicted with less accuracy overall, but one significant deviation is observed in the stress σ_{xz}. The discrepancy between the predicted and reference values of σ_{xz} is evident in figure 4.24(g) and 4.24(h), respectively. This substantial deviation in different shear stresses impacts the calculation of von Mises stress, as illustrated in figure 4.24(j), where the stress does not appear concentrated at the top and bottom edges of the fixed surfaces as shown in the reference solution, figure 4.24(k).

The observed discrepancies may stem from imbalances in loss function weighting. Despite efforts to adjust loss weights to equalize the learning of different output parameters, certain outputs continue to exhibit poor prediction performance. For instance, assigning lower weights to high-loss parameters like σ_{zz} and higher weights to others did not yield the desired improvements in predictive accuracy. The results, encompassing predicted displacements, stresses, and von Mises stress, are presented comprehensively. Addressing these challenges requires further investigation into refining the PINN architecture, optimizing loss function strategies, and exploring alternative training methodologies to achieve

more consistent and accurate predictions across all output parameters.

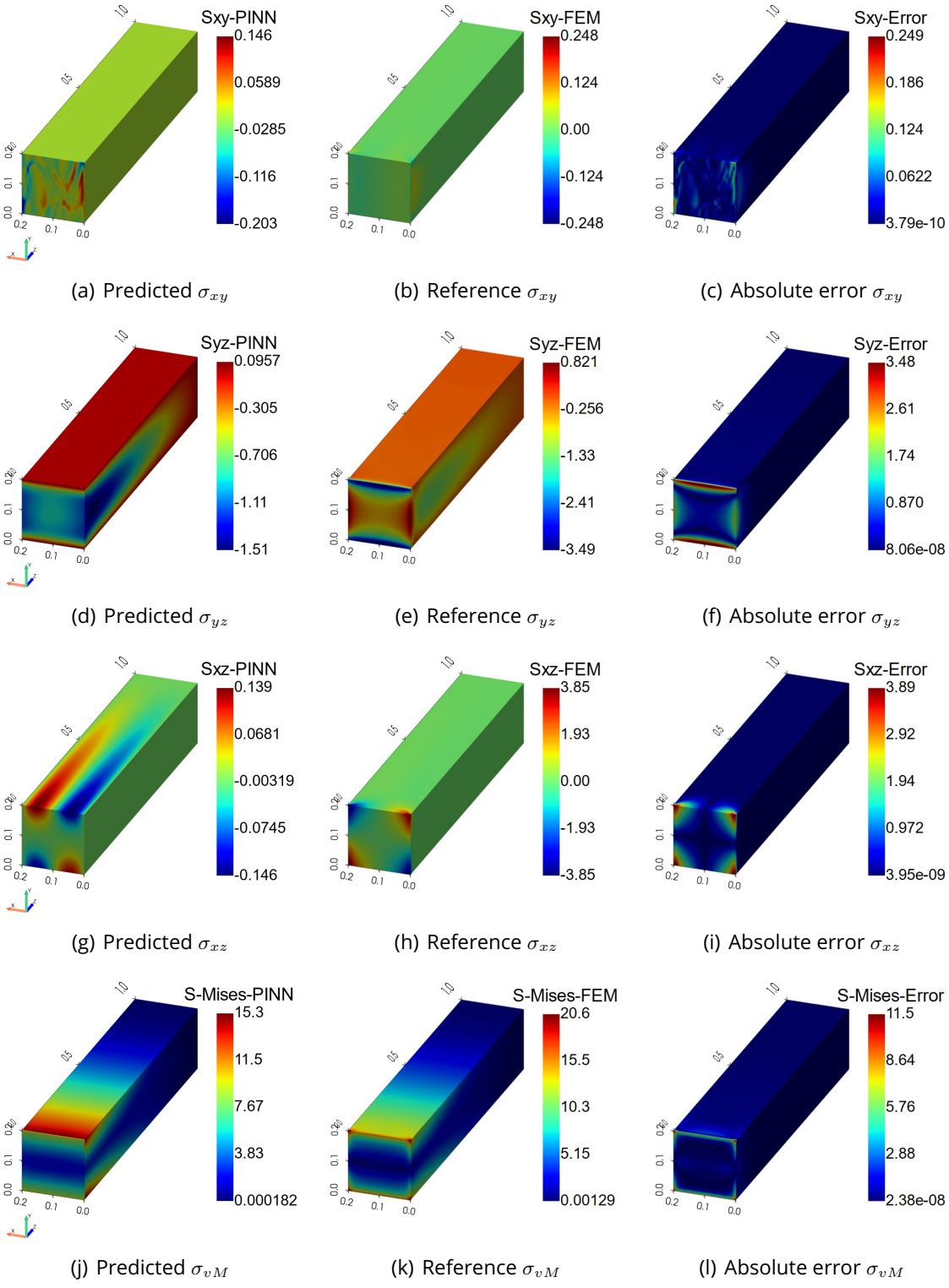


(a) Predicted σ_{xx} (b) Reference σ_{xx} (c) Absolute error σ_{xx}

(d) Predicted σ_{yy} (e) Reference σ_{yy} (f) Absolute error σ_{yy}

(g) Predicted σ_{zz} (h) Reference σ_{zz} (i) Absolute error σ_{zz}

Figure 4.23: 3D Cantilever beam problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of σ_{xx}, σ_{yy} and σ_{zz} respectively



(a) Predicted σ_{xy} (b) Reference σ_{xy} (c) Absolute error σ_{xy}

(d) Predicted σ_{yz} (e) Reference σ_{yz} (f) Absolute error σ_{yz}

(g) Predicted σ_{xz} (h) Reference σ_{xz} (i) Absolute error σ_{xz}

(j) Predicted σ_{vM} (k) Reference σ_{vM} (l) Absolute error σ_{vM}

Figure 4.24: 3D Cantilever problem: (a), (d), (g) are the predicted solutions; (b), (e), (h) are the reference solutions; (c), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of σ_{xy}, σ_{yz}, σ_{zx}, and σ_{vM} respectively

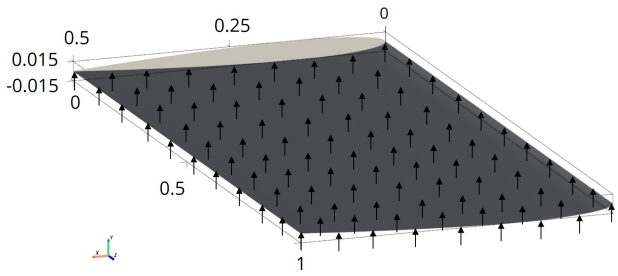
4.2.3 3D airfoil Subjected to Lift Forces

In this subsection, we will examine the application of lift forces on a simplified representation of an aircraft wing, specifically a 3D airfoil. The airfoil profile is chosen arbitrarily as "NACA 0006" [12], depicted in figure 4.25(a), is anchored at one face along z = 0 and free at the opposite end. As elucidated in Section 3.3.1, the computation of lift forces on the airfoil is achieved through the vortex lattice method with the following parameters:

Table 4.4: Details on airfoil profile and VLM parameters

Profile parameters	VLM parameters
length of chord = 0.5 m	airplane velocity = 25 m/s
length of wing span = 1 m	angle of attack = 5
angle of twist = 0	-

These lift forces are utilized to ascertain traction forces, denoted as force per unit area, on the underside of the airfoil, thereby establishing traction boundary conditions, as illustrated in figure 4.25. The table in figure 4.25(b) outlines the boundary conditions across distinct boundaries. The lift forces are exerted solely in the vertical direction, thus resulting in a traction component along the vertical axis, $\bar{t} = (\mathbf{0}\ \ \mathbf{t_y}\ \ \mathbf{0})$, while the remaining components are nullified.



(a)

Boundary	Condition
z = 0	$u_x = 0,\ u_y = 0,\ u_z = 0$
x = 0.5	$\sigma_{xx} = 0,\ \sigma_{xy} = 0,\ \sigma_{xz} = 0$
z = 1	$\sigma_{zz} = 0,\ \sigma_{zx} = 0,\ \sigma_{zy} = 0$
top	$\sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z = 0,$ $\sigma_{yx}n_x + \sigma_{yy}n_y + \sigma_{yz}n_z = 0,$ $\sigma_{zx}n_x + \sigma_{zy}n_y + \sigma_{zz}n_z = 0,$
bottom	$\sigma_{xx}n_x + \sigma_{xy}n_y + \sigma_{xz}n_z = 0,$ $\sigma_{yx}n_x + \sigma_{yy}n_y + \sigma_{yz}n_z = t_y,$ $\sigma_{zx}n_x + \sigma_{zy}n_y + \sigma_{zz}n_z = 0,$

(b)

Figure 4.25: (a) 3D airfoil subjected to lift forces implemented as traction, (b) Displacement and traction conditions on different boundaries. The material parameters are: E = 10^8 Pa, $\nu = 0.3$

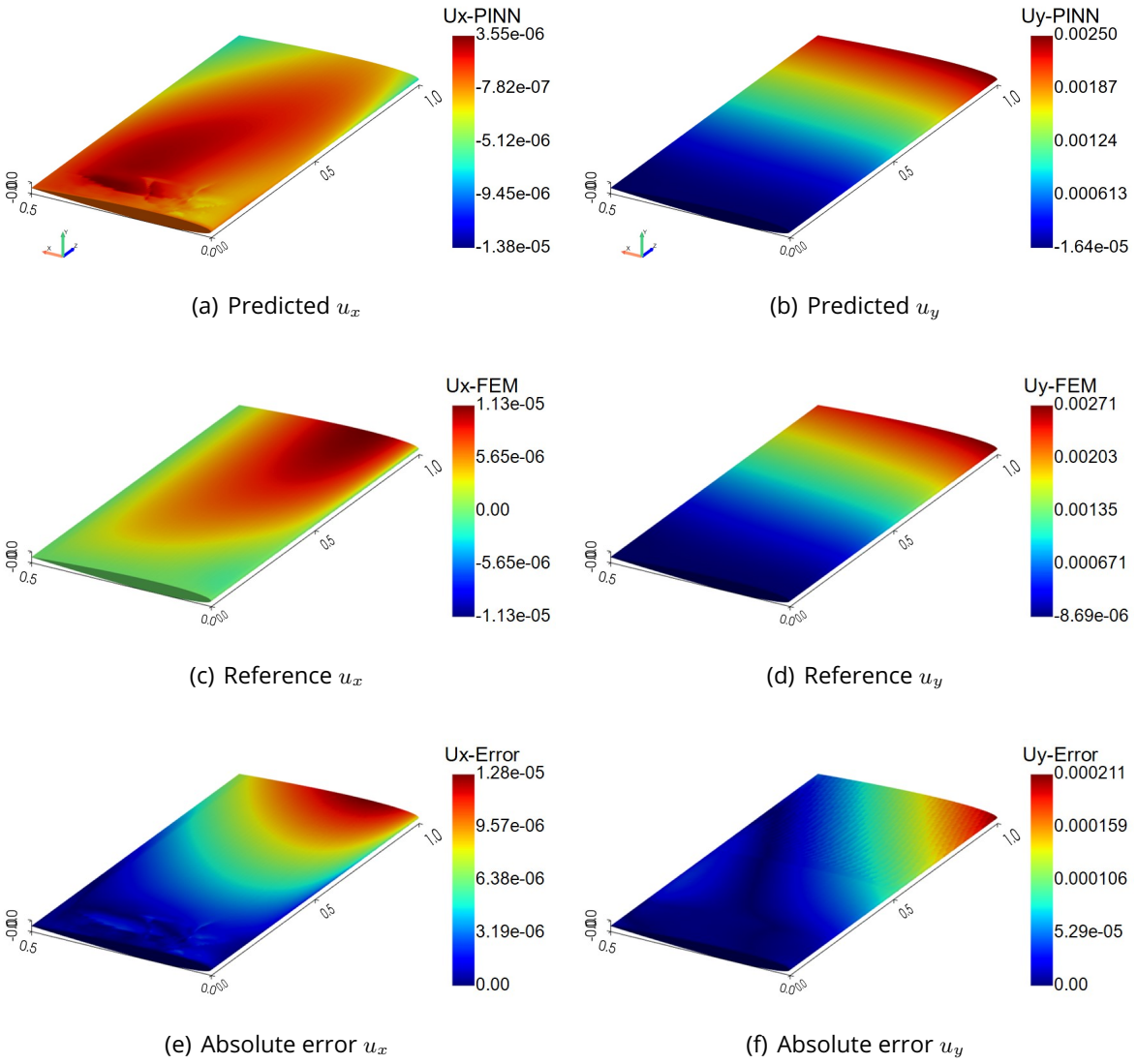
Navigating the complexities of output solutions across a broad spectrum poses a challenge

for PINN architectures to predict results accurately. As demonstrated earlier, achieving accurate solutions often requires a densely populated neural network with many neurons and layers. In this case, the predicted solutions were notably unlike the reference solutions, with unusual distributions that did not align with expected patterns. This discrepancy could be attributed to the limitations of the PINN model itself, but it's also important to consider the possibility of improper problem implementation contributing to these challenges.

Alternatively, the PINN model was trained using known data imported from FEM software, with each output solution supplemented by 50 known data points (simulation data). However, the efficacy of the PINN model in learning solutions could be compromised if these points were randomly selected. To address this, a systematic distribution of points was adopted. Primarily, an abundance of points was randomly dispersed along the fixed surface, where stresses are pronounced and exhibit intricate nonlinear distribution. By augmenting points along the fixed boundary, the PINN model's capacity to comprehend solutions effectively was enhanced. Additionally, some points were allocated along the free end at $z = 1$, facilitating the observation of displacement distributions. Finally, a sparse distribution of points was implemented near the fixed boundary, with a small number of points randomly scattered within the domain.

The PINN model for this airfoil configuration was trained with 160 neurons and 6 layers, employing 9 MLPs for each output parameter. The training process involved around 27,000 training points, and 2 million iterations, taking approximately 192 hours of training time. The calculated traction forces were on the order of 100 N/m² near the leading edge of the airfoil, which led to the selection of a large Young's modulus (E) value for a meaningful simulation.

The figures 4.26 and 4.27 present the predicted output solution, reference solution, and absolute error for different outputs. The concentration of stress on the fixed surface can be attributed to the boundary conditions, which impose constraints on the deformation of the structure. The PINN model demonstrated reasonable convergence in the results, although it did not achieve accurate predictions compared to the reference solution. Notably, the von Mises stress prediction in figure 4.27(b) showed relatively good convergence. However, there were significant deviations observed in some stress components, particularly in the shear stresses. The observed deviations highlight areas for further refinement and optimization of the model.



(a) Predicted u_x (b) Predicted u_y

(c) Reference u_x (d) Reference u_y

(e) Absolute error u_x (f) Absolute error u_y

Figure 4.26: 3D airfoil problem: (a), (b) are the predicted solutions; (c), (d) are the reference solutions; (e), (f) are the absolute differences between the reference and predicted solutions, for the value of u_x and u_y respectively

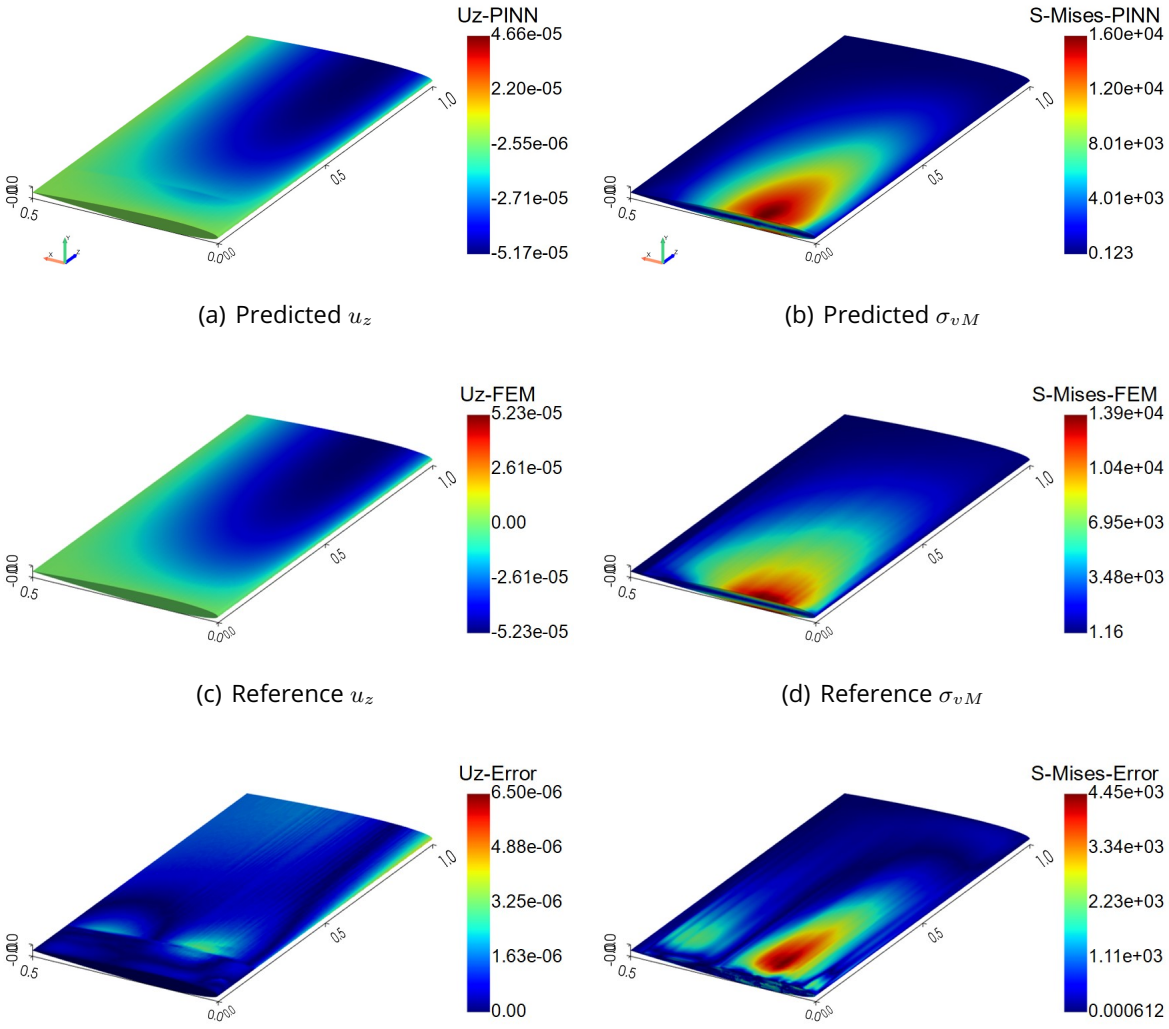
The displacement prediction is in good accuracy, for output u_y and u_z, and significant deviation in u_x as shown in figure 4.26(a). This is likely due to not sufficient known data points near the free boundary surface. Again the error is more for all the displacements near the boundary with maximum value as shown in figure 4.26(e), 4.26(f), 4.27(e) for u_x, u_y and u_z respectively. Training with a higher number of neurons and utilizing more data points could yield improved results.

During the analysis, a notable observation was made regarding the training process using known data points. The reference stress values were of higher magnitudes, on the order of

10^4, while displacements were relatively lower, around 10^{-5}. This disparity in magnitudes led to corresponding differences in the losses associated with stress and displacement. Despite applying lower weights to observed stress data and higher weights to observed displacement data, the model learned stress values more rapidly compared to displacements. The loss associated with displacements was considerably lower, approximately 10^{-8}, whereas, for stress, it was around 10^{-1} which reduced the importance of displacements and gave more importance to stresses. This imbalance in learning emphasized the importance of properly weighing the losses of different parameters. In this case, providing substantially larger weights to displacement data could enhance the model's focus on learning displacement values more effectively. By adjusting the weighting scheme, the training process can be optimized to prioritize the accurate learning of both stress and displacement components, leading to improved overall performance and convergence of the model.

Training the model with a larger dataset could yield more accurate results. However, the primary focus of this study was to predict solutions without relying extensively on preexisting data, although achieving this directly was not possible, necessitating the use of observed data. Training with a substantial dataset is commonly employed for material parameter identification, addressing inverse problems, which has shown success in various applications. It is recognized that the vanilla PINN method may be more effective for inverse problems compared to forward problems [62] [40], as evidenced by the challenging results encountered in this study. Nevertheless, the research aims to develop and enhance convergence specifically for forward problems, acknowledging the limitations of the vanilla PINN approach in this context.

The forthcoming section will discuss potential solutions and new methods aimed at addressing the convergence challenges encountered in forward problems, offering insights into advancing the effectiveness of PINN models for such applications.



(a) Predicted u_z

(b) Predicted σ_{vM}

(c) Reference u_z

(d) Reference σ_{vM}

(e) Absolute error u_z

(f) Absolute error σ_{vM}

Figure 4.27: 3D airfoil problem: (a), (b) are the predicted solutions; (c), (d) are the reference solutions; (e), (f), (i) are the absolute differences between the reference and predicted solutions, for the value of u_z, and σ_{vM} respectively

4.3 Conclusion and Further Research

In this chapter, we explored the training of the PINN model to predict solutions for a range of problems, encompassing simple to complex geometries, varied force applications, and intricate boundary conditions. The outcomes highlight key aspects of PINN training, including the challenges posed by problem complexity, convergence difficulties, and model performance. Moving from 2D to 3D linear elasticity analyses introduces a multitude of equations, increasing the number of PDE residuals and boundary condition residuals, resulting in a more extensive set of loss terms that can hinder model convergence and optimization. Additionally, the nature of constraints and boundary conditions significantly impacts the convergence process, as demonstrated in Section 4.1.4.

The evaluation of results revealed that training PINNs for 3D geometries in the context of linear elasticity equations posed significant challenges. While the PINN model accurately predicted solutions for 2D geometries with varying boundary conditions and force applications, achieving similar results in 3D scenarios proved more demanding. Section 4.2.1 exemplifies this challenge, where the PINN model accurately predicted solutions for a simple 3D cube with basic boundary constraints but struggled with more complex scenarios involving fixed constraints.

Based on the observations, training the PINN model for 3D geometries has proven to be challenging. To enhance performance, improvements in optimization methods, utilization of better loss functions, employment of deeper neural networks with increased neuron counts and layers, or even a complete redesign of the neural network architecture could be explored.

In recent research by Vasiliy A. Es'kin [26], it was demonstrated that solving the 3D cantilever beam problem with a more densely structured neural network is feasible. Their model utilized 9 MLPs, each consisting of 512 neurons and 6 layers, trained for 2 million iterations. However, implementing such a large model was constrained by resource limitations in our case, particularly in terms of memory requirements.

Es'kin's work introduces the Separable Physics-Informed Neural Network (SPINN) architecture, which utilizes the classic separation of variables method (Fourier method) for neural network representation. SPINN adopts a per-axis basis instead of point-wise processing, reducing the number of network forward passes needed. This approach significantly de-

creases training and evaluation times, as well as memory costs, compared to conventional PINNs. SPINN was reported to be nearly 70 times faster than traditional PINNs for solving their specific problem. Moreover, Es'kin et al. demonstrated the use of the Direct Energy method for implementing the loss function instead of relying solely on PDE residuals, which further improved convergence rates. Additionally, SPINN's memory usage is notably lower compared to PINNs.

While SPINN represents a significant advancement in PINN methodologies, its implementation was beyond the scope of this thesis due to time constraints. Nonetheless, this research showcases a promising direction for PINNs, offering convergence and computational efficiency comparable to traditional FEM approaches.

5 Summary and Outlook

PDEs are essential for describing physical phenomena, but solving them with complex models is challenging. Machine learning, particularly using artificial neural networks, offers a novel approach to predicting solutions based on specific constraints and boundary conditions, providing an alternative to traditional numerical methods.

This thesis explored the application of PINNs for predicting solutions of PDEs. The second chapter introduced artificial neural networks and discussed their integration with physics-informed components. Key topics included neural network architecture, parameters, and their adaptation to physics-based learning. PINNs are highlighted for their use of physical equations and specific boundary conditions to guide training. Optimization methods, particularly the ADAM optimizer, were discussed for achieving convergence in solution predictions, with a focus on linear elasticity equations in 2D and 3D settings. The study showcases how boundary conditions are essential for defining and solving these physical systems effectively, emphasizing the role of accurate physics modeling and robust optimization in achieving reliable predictive outcomes.

In the third chapter, various methodologies used in this study were explored. Notably, employing multiple MLPs for each output was discussed to improve solution accuracy. The study utilized the DeepXDE library, enabling efficient solving of PDEs across different models. Additionally, the method to map forces from CFD tools into FEM software and the PINN model was outlined, showcasing how aerodynamic forces can be incorporated into structural simulations using the AeroSandBox library.

In the fourth chapter, extensive discussions were centered around the results obtained from the PINN model. Initially, the training of the PINN model for two dimensional geometries demonstrated its capability to predict solutions across varying scenarios accurately. Different geometries were subjected to diverse force applications and boundary conditions to analyze their effects on the model's performance. An important finding highlighted was the significant influence of boundary conditions on the convergence process. It became evident that stricter and more complex or nonlinear outputs led to slower convergence rates. This difficulty in training the model resulted in longer computation times, ultimately impacting the model's overall performance. The chapter emphasized the nuanced relationship between model complexity, boundary conditions, convergence behavior, and computational efficiency. Exploring a cantilever beam under uniform traction revealed conver-

gence challenges with PINNs, attributed to output complexity and boundary conditions, necessitating extensive training iterations for accuracy. Examining boundary conditions, roller boundary conditions showed faster convergence than fixed boundary conditions in an annular quarter disk geometry, highlighting the impact of boundary constraints on optimization processes.

Training the PINN model for three-dimensional linear elasticity problems demanded extensive computational resources and careful parameter tuning due to the complexity of the PDEs and boundary conditions. Despite these challenges, the PINN model demonstrated promising predictive capabilities, especially in predicting displacements. Analyzing specific case studies, such as the cubic structure with a spherical void and the 3D cylindrical and cantilever beam geometries, highlighted the difficulties in achieving accurate predictions for the stresses, particularly near fixed boundaries. The fixed boundary conditions had a notable impact on convergence rates and solution accuracy, highlighting the need for improved PINN architectures and optimization methods for better performance. Additionally, when applied to a 3D airfoil geometry, the PINN model produced suboptimal results, necessitating the use of known data to train the model and improve solution accuracy.

Looking ahead, advancements in PINN methodologies, such as exploring denser MLPs or adopting different neural network architectures like the Separable Physics-Informed Neural Network (SPINN), show promising prospects for enhancing convergence rates and computational efficiency. SPINN's utilization of per-axis basis representation and the Direct Energy method for implementing loss functions offer a viable approach to tackle challenges associated with three-dimensional linear elasticity analysis. Future research should prioritize further development of PINN methodologies to enhance their capability in accurately and efficiently modeling complex physical systems. This involves refining convergence rates and optimizing computational efficiency to broaden the scope of applications in engineering and scientific domains.

Bibliography

[1] 3d vortex lattice method. [Accessed on April 25, 2024] Available at: https://www.aerodynamics4students.com/subsonic-aerofoil-and-wing-theory/3d-vortex-lattice-method.php.

[2] Activation function. [Accessed on Feb 25, 2024] Available at: https://en.wikipedia.org/wiki/Activation_function.

[3] Aircraft wing types and classifications. [Accessed on April 12, 2024] Available at: https://pilotinstitute.com/aircraft-classification/.

[4] Biological neuron model. [Accessed on March 06, 2024] Available at: https://en.wikipedia.org/wiki/Biological_neuron_model.

[5] Biot–savart law. [Accessed on April 22, 2024] Available at: https://en.wikipedia.org/wiki/Biot%E2%80%93Savart_law.

[6] Center for information services and high performance computing. https://tu-dresden.de/zih/hochleistungsrechnen/nhr-center.

[7] Diffusion equation. [Accessed on Feb 29, 2024] Available at: https://en.wikipedia.org/wiki/Diffusion_equation.

[8] Laplace operator, encyclopedia of mathematics. [Accessed on April 19, 2024] Available at: http://encyclopediaofmath.org/index.php?title=Laplace_operator&oldid=51994.

[9] Learning rate. [Accessed on Feb 25, 2024] Available at: https://en.wikipedia.org/wiki/Learning_rate.

[10] List of optimization algorithms. [Accessed on April 12, 2024] Available at: https://en.wikipedia.org/wiki/List_of_optimization_algorithms.

[11] Mcculloch-pitts neuron — mankind's first mathematical model of a biological neuron, 2018. [Accessed on March 06, 2024] Available at: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

[12] Uiuc airfoil coordinate database. [Accessed on May 10, 2024] Available at: https://m-selig.ae.illinois.edu/ads/coord_database.html.

[13] I. Abd El Kader, G. Xu, Z. Shuai, S. Saminu, I. Javaid, and I. Salim Ahmad. Differen-

tial deep convolutional neural network model for brain tumor classification. *Brain Sciences*, 11, 2021.

[14] R. Abdulkadirov, P. Lyakhov, and N. Nagornov. Survey of optimization algorithms in modern neural networks. *Mathematics*, 11(11), 2023.

[15] C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer International Publishing AG, Gewerbestrasse 11, 6330 Cham, Switzerland, 2018.

[16] G. Andrew and J. Gao. Scalable training of l1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 33–40, New York, NY, USA, 2007. Association for Computing Machinery.

[17] J. Bai, T. Rabczuk, A. Gupta, L. Alzubaidi, and Y. Gu. A physics-informed neural network technique based on a modified loss function for computational 2d and 3d solid mechanics. *Computational Mechanics*, 71:543–562, 2023.

[18] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:1–43, 2018.

[19] H. Brezis and F. Browder. Partial differential equations in the 20th century. *Advances in Mathematics*, 135(1):76–144, 1998.

[20] S. L. Brunton and J. N. Kutz. Machine learning for partial differential equations, 2023.

[21] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. MPS-SIAM Series on Optimization, SIAM, Philadelphia, 2009.

[22] S. Cuomo, V. S. D. Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, pages 92–88, 2022.

[23] M. Drela. *Flight Vehicle Aerodynamics*. MIT Press, Cambridge, MA, 2014.

[24] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[25] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[26] V. A. Es'kin, D. V. Davydov, J. V. Gur'eva, A. O. Malkhanov, and M. E. Smorkalov. Separable physics-informed neural networks for the solution of elasticity problems, 2024.

[27] J. H. Ferziger, M. Perić, and R. L. Street. *Solution of the Navier–Stokes Equations: Part 1*, pages 183–226. Springer International Publishing, Cham, 2020.

[28] C. Gallo. *Encyclopedia of Information Science and Technology*, volume pp. 6369-6378. IGI Global, 2015.

[29] R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and J. M. Z. Maningo. Object detection using convolutional neural networks. In *TENCON 2018 - 2018 IEEE Region 10 Conference*, pages 2023–2027, 2018.

[30] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[31] G. W. Griffiths and W. E. Schiesser. Linear and nonlinear waves. *Scholarpedia*, 4(7):4308, 2009. revision #154041.

[32] G. E. M. G.Thomas Mase. *Continuum Mechanics For Engineers*. CRC Press LLC, 1999.

[33] Q. He, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *Advances in Water Resources*, 141:103610, 2020.

[34] P. Jain and P. Kar. Non-convex optimization for machine learning. *Foundations and Trends® in Machine Learning*, 10(3–4):142–336, 2017.

[35] S. J. W. Jorge Nocedal. *Numerical Optimization*, volume 2. Springer New York, NY, 2006.

[36] B. Khemani, S. Patil, K. Kotecha, and S. Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11, 2024.

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[38] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[39] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[40] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43:B1105–B1132, 2020.

[41] D. J. Lucia, P. S. Beran, and W. A. Silva. Reduced-order modeling: new approaches for computational physics. *Progress in Aerospace Sciences*, 40(1):51–117, 2004.

[42] V. MA and T. MD. A variational bayesian neural network for structural health monitoring and cost-informed decision-making in miter gates. *Structural Health Monitoring*, 21:4–18, 2022.

[43] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conference on Natural Language Learning - Volume 20*, COLING-02, page 1–7, USA, 2002. Association for Computational Linguistics.

[44] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[45] C. Morris. Graph neural networks: Graph classification. In L. Wu, P. Cui, J. Pei, and L. Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 179–193. Springer Singapore, Singapore, 2022.

[46] V. M. Nguyen-Thanh, C. Anitescu, N. Alajlan, T. Rabczuk, and X. Zhuang. Parametric deep energy approach for elasticity accounting for strain gradient effects. *Computer Methods in Applied Mechanics and Engineering*, 386:114096, 2021.

[47] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 873–880, New York, NY, USA, 2009. Association for Computing Machinery.

[48] H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.

[49] S. Ruder. An overview of gradient descent optimization algorithms, 2017.

[50] N. Sharma, V. Jain, and A. Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018. International Conference on Computational Intelligence and Data Science.

[51] P. D. Sharpe. Aerosandbox: A differentiable framework for aircraft design optimization. Master's thesis, Massachusetts Institute of Technology, 2021.

[52] M. V. Shivaditya, J. Alves, F. Bugiotti, and F. Magoules. Graph neural network-based

surrogate models for finite element analysis, 2022.

[53] C. Stover and E. W. Weisstein. Closed-form solution-from mathworld–a wolfram web resource. https://mathworld.wolfram.com/closed-formsolution.html.

[54] F. Sultana, A. Sufian, and P. Dutta. Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, 201-202:106062, 2020.

[55] L. Sun, H. Gao, S. Pan, and J.-X. Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[56] S. Sun, Z. Cao, H. Zhu, and J. Zhao. A survey of optimization methods from a machine learning perspective, 2019.

[57] J. Tang and R. Liao. Graph neural networks for node classification. In L. Wu, P. Cui, J. Pei, and L. Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 41–61. Springer Singapore, Singapore, 2022.

[58] Y. Tian, Y. Zhang, and H. Zhang. Recent advances in stochastic gradient descent in deep learning. *Mathematics*, 11(3), 2023.

[59] T. Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

[60] N. S. Wadia, Y. Dandi, and M. I. Jordan. A gentle introduction to gradient-based optimization and variational inequalities for machine learning, 2024.

[61] J. Wang and B. Malakooti. A feedforward neural network for multiple criteria decision making. *Computers Operations Research*, 19:151–167, 1992.

[62] W. Wu, M. Daneker, M. A. Jolley, K. T. Turner, and L. Lu. Effective data sampling strategies and boundary condition constraints of physics-informed neural networks for identifying material properties in solid mechanics, 2022.

[63] X.-S. Yang. *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley Sons, Inc., John Wiley Sons, Inc., Hoboken, New Jersey, Canada, 2010.

[64] M. Zhang. Graph neural networks: Link prediction. In L. Wu, P. Cui, J. Pei, and L. Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 195–223. Springer Singapore, Singapore, 2022.

List of Figures

List of Tables

Appendix

A.1 Optimization Algorithm: ADAM

Algorithm 2 Adam: Stochastic Optimization Algorithm [37]

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector

$m_0 \leftarrow 0$ ▷ Initialize 1st moment vector
$v_0 \leftarrow 0$ ▷ Initialize 2nd moment vector
$t \leftarrow 0$ ▷ Initialize timestep
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get gradients w.r.t. stochastic objective at timestep t
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second raw moment estimate
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ Compute bias-corrected first moment estimate
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ Compute bias-corrected second raw moment estimate
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ ▷ Update parameters
end while
return θ_t ▷ Resulting parameters

A.2 PINN Model Parameters

Table A.1 represents different parameters of the PINN model utilized for different cases. Here layers are represented as follows: [20×5], represents an MLP with 20 neurons and 5 Layers, and [20 × 5] × 5, represents 5 such MLPs for 5 outputs in case of 2D problems, similarly 9 outputs in case of 3D problems. The training time is represented in hours followed by minutes (hours: minutes).

Table A.1: PINN Model Parameters for different cases

Model	Training points	Iterations	Training time (hrs)	Layers
Pure Bending Beam (4.1.1)	1500	12,000	0:27	[20 × 5] × 5
Thin Perforated Plate (4.1.2)	5000	30,000	1:09	[30 × 5] × 5
2D Cantilever Beam (4.1.3)	7000	1,500,000	22:17	[40 × 5] × 5
Disk - Roller BC (4.1.4)	6000	30,000	1:35	[40 × 5] × 5
Disk - Fixed BC (4.1.4)	6000	1,300,000	18:05	[48 × 5] × 5
3D Cube (4.2.1)	8814	550,000	21:02	[30 × 6] × 9
3D Cylinder 4.2.2	27,000	2,200,000	257:08	[160 × 6] × 9
3D Cantilever Beam (4.2.2)	27,000	1,800,000	190:80	[160 × 6] × 9
3D Airfoil (4.2.3)	27,000	2,000,000	192:26	[160 × 6] × 9