PHYSICS INFORMED NEURAL NETWORKS FOR COMPUTATIONAL FLUID DYNAMICS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ATAKAN AYGÜN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

JANUARY 2023

Approval of the thesis:

PHYSICS INFORMED NEURAL NETWORKS FOR COMPUTATIONAL FLUID DYNAMICS

submitted by **ATAKAN AYGÜN** in partial fulfillment of the requirements for the degree of **Master of Science in Mechanical Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. M. A. Sahir Arıkan
Head of Department, **Mechanical Engineering** _____

Assist. Prof. Dr. Ali Karakuş
Supervisor, **Mechanical Engineering, METU** _____

Dr. Romit Maulik
Co-supervisor, **Mathematics and Computer Science, Argonne National Laboratory** _____

Examining Committee Members:

Prof. Dr. Cüneyt Sert
Mechanical Engineering, METU _____

Assist. Prof. Dr. Ali Karakuş
Mechanical Engineering, METU _____

Assist. Prof. Dr. Özgür Uğraş Baran
Mechanical Engineering, METU _____

Assist. Prof. Dr. Altuğ Özçelikkale
Mechanical Engineering, METU _____

Assoc. Prof. Dr. Barbaros Çetin
Mechanical Engineering, Bilkent University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Atakan Aygün

Signature :

ABSTRACT

PHYSICS INFORMED NEURAL NETWORKS FOR COMPUTATIONAL FLUID DYNAMICS

Aygün, Atakan
M.S., Department of Mechanical Engineering
Supervisor: Assist. Prof. Dr. Ali Karakuş
Co-Supervisor: Dr. Romit Maulik

January 2023, 81 pages

In this work, we use physics-informed neural networks (PINN) to model the problems generally used in computational fluid dynamics (CFD). Since PINN does not need any discretization scheme or mesh generation, this approach is easier to implement compared to conventional numerical methods. In the context of CFD problems, the effect of network parameters is shown in the convergence and the accuracy of the solution. The solutions to forward problems are chosen with simple geometries since as the nonlinearity increases in the PDE, the PINN has difficulties providing physically meaningful solutions. Therefore, the provided solutions to flow problems are in low Reynolds number regions. Applications in this thesis include the solution of flow problems represented with Navier-Stokes and Euler equations. The problems in Euler equations are in a one-dimensional domain. The solution of thermal convection equations that couples the flow equations with the energy equation is presented. The effect of weighting in each loss term is presented along with different neural network types. Mesh deformation for moving boundary problems in CFD is presented. The deformation is modeled with elliptic equations. The exact boundary values are

enforced on the network output to ensure the boundary is in its exact position. The quality of the mesh is presented with the element shape and area changes. PINN can be used in sub-tasks that do not affect the accuracy of high-fidelity solvers.

Keywords: physics informed neural networks, computational fluid dynamics, thermal convection, mesh deformation

ÖZ

HESAPLAMALI AKIŞKANLAR DİNAMİĞİNDE FİZİKLE ÖĞRENEN YAPAY SİNİR AĞLARI

Aygün, Atakan
Yüksek Lisans, Makina Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Ali Karakuş
Ortak Tez Yöneticisi: Dr. Romit Maulik

Ocak 2023 , 81 sayfa

Bu çalışmada, genellikle hesaplamalı akışkanlar dinamiğinde (CFD) kullanılan problemleri modellemek için fizikle öğrenen yapay sinir ağlarını (PINN) kullanıyoruz. PINN herhangi bir ayrıklaştırma şemasına veya ağ oluşturmaya ihtiyaç duymadığından, bu yaklaşımın uygulanması geleneksel sayısal yöntemlere kıyasla daha kolaydır. CFD problemleri bağlamında, ağ parametrelerinin etkisi, çözümün yakınsama ve doğruluğunda gösterilmiştir. PDE'de doğrusal olmama arttıkça, PINN fiziksel olarak anlamlı çözümler sağlamakta zorluk çektiğinden, doğrudan problemlere yönelik çözümler karmaşık olmayan bölgelerde seçilir. Bu nedenle akış problemlerine sağlanan çözümler düşük Reynolds sayılı bölgelerdedir. Uygulamalar Navier-Stokes ve Euler denklemleriyle temsil edilen akış problemlerinin çözümünü içermektedir. Euler denklemlerindeki problemler, mevcut PINN yetenekleri bölgesinde, tek boyutlu bir alandadır. Akış denklemlerini enerji denklemiyle birleştiren ısıl taşınım denklemlerinin çözümü sunulmaktadır. Her kayıp terimindeki ağırlıklandırmanın etkisi, farklı sinir ağı türleri ile birlikte sunulur. Son olarak CFD'de hareketli sınır problemleri için ağ

deformasyonu sunulmuştur. Deformasyon eliptik denklemlerle modellenmiştir. Kesin sınır değerleri, sınırın gerçek konumunda olmasını sağlamak için ağ çıkışına uygulanır. Ağın kalitesi, eleman şekli ve alan değişiklikleri ile sunulur.

Anahtar Kelimeler: fizikle öğrenen yapay sinir ağları, hesaplamalı akışkanlar dinamiği, ısıl taşınım, ağ bozunumu

To my beloved family.

ACKNOWLEDGMENTS

Firstly I would like to express my sincere gratitude and respect to my advisors Dr. Romit Maulik and Dr. Ali Karakuş for their constant support and guidance for this thesis. Although Dr. Karakuş's immense knowledge of numerical methods left me shaken during our coffee breaks, I learned more than I could imagine.

I am very grateful to be a member of a research group. I would like to thank all the members of the Accelerated Multiphysics Research Group. Having people around with similar interests always helps me to motivate and improve myself. Thanks to them for their support in my struggles and fun conversations in our laboratory.

I would like to thank my friends who look at me and think doing an MSc. is easy. They show me constant support and bring me up when I am down. They were amazing throughout this journey even though generally they did not understand what I am doing. And of course, I am very grateful that they are always hungry for a 2 a.m. snack.

Last, but the most, I am very grateful to my parents and my brother. During the pandemic, we faced some challenges as a family. Together we overcome the struggle. I am deeply thankful for their continuous support, unconditional love, and financial support during my unemployment period. Without them, this research would never have been completed.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
PDE	Partial Differential Equation
PINN	Physics Informed Neural Network
FDM	Finite Difference Method
FVM	Finite Volume Method
FEM	Finite Element Method
DG	Discontinuous Galerkin
ANN	Artificial Neural Network
ODE	Ordinary Differential Equation
NS	Navier-Stokes
MLP	Multilayer Perceptron
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
BGK	Bhatnagar–Gross–Krook
cPINN	Conservative Physics Informed Neural Network
XPINN	Extended Physics Informed Neural Network
VPINN	Variational Physics Informed Neural Network
BFGS	Broyden–Fletcher–Goldfarb–Shanno
NTK	Neural Tangent Kernel
SPH	Smoothed Particle Hydrodynamics
DEM	Diffuse Element Method

CHAPTER 1

INTRODUCTION

In the first part of this chapter, the literature survey is presented to show the current state of the art. The solution methods for partial differential equations (PDE) are presented in the first section, including the conventional methods and machine learning approaches. Followed by the introduction of physics-informed neural networks (PINN). The current work on the PINN methodology and its applications are presented. In the second part, the motivation of this thesis is explained.

1.1 Overview of Conventional Differential Equation Solvers

Differential equations arise in many engineering problems, such as fluid flow, heat transfer, wave propagation, etc. Solving these equations is important to represent these physics in real life. To do so, there are quite a number of different methods. Conventional numerical methods such as the finite difference method (FDM), finite volume method (FVM), and finite element methods (FEM) are widely used. In the finite difference method, which is the historically oldest method [1], a grid is constructed in space. The geometry is represented with K finite points on the computational domain. In the neighborhood of each grid point x^k, the solution is approximated by local polynomials. These K finite points yield K finite difference equations for K unknowns. To approximate the first and second derivatives, which arise in conservation laws, Taylor series expansion or polynomial fitting is generally used. This method is appealing due to its simplicity. Especially on structured grids, the finite difference method is very effective [4]. However, Since FDM relies on a local one-dimensional polynomial approximation, it struggles in higher dimensions.

Also, the method has complications around the boundaries and discontinuous regions. Due to these problems, FDM is ill-suited for complex geometries in the aspect of both discontinuous regions and the local order of the solution [1].

These drawbacks of finite difference methods lead to methods that have geometric flexibility. To overcome these, element-based methods are introduced. These methods partition the overall domain Ω into k different elements, D^k, generally triangles or quadrilaterals in two-dimension, and tetrahedra or hexahedra in three-dimension. One is the finite volume method, in which the solution is subdivided into a finite number of control volumes. In the simplest form, the solution is approximated at the centroid of each control volume by a constant. In FVM, the approximation is purely local, and therefore, it has no condition on the grid structure. The conservation laws are solved in the control volumes. This yields an algebraic equation for each control volume with the number of neighboring nodes appearing. The flux in FVM is evaluated in the boundary of the element. However, the solution is approximated in the cell center. There are several ways to approximate these fluxes and details lead to different FVM [1]. These terms reduce a surface term by using the divergence theorem, and these integrals are approximated by suitable quadrature methods [4]. In FVM, problems occur if one wants to increase the order of the accuracy. Increasing the order introduces a need to have a specific mesh structure. This opposes the initial need to have geometric flexibility. The high-order reconstruction with proper methods in FVM is both complex and prone to instability [1]. Thus, the main limitation of FVM is extending to higher order accuracy on general unstructured meshes.

The finite element method distinguishes itself from the FVM by multiplying the equations by a weight function before the integration over the entire domain [4]. The solution is approximated by a shape function within an element. The simplest approach introduces linear shape functions that guarantee the continuity of the solution across the boundaries [4]. The main idea to introduce another approach beyond the FVM is to obtain high-order approximations. This can be achieved in FEM by introducing additional degrees of freedom inside an element. This allows having different orders of approximation on each element, enabling changes in both size and order named as hp-adaptivity [1]. Using symmetric basis functions in the classical finite element approach can be unstable for some problems such as wave problems and conservation

Table 1.1: Generic properties of common numerical methods. + and x indicate success and failure respectively, while the (+) symbol indicates that the method requires modifications to be capable of solving the problem [1].

	FDM	FVM	FEM	DG
Complex geometries	x	+	+	+
High-order accuracy	+	x	+	+
Explicit semi-discrete form	+	+	x	+
Wave dominated problems	+	+	(+)	+
Elliptic problems	+	(+)	+	(+)

laws. In these types of problems, the information propagates from a specific direction. In FDM and FVM this problem can be solved by using an upwinding scheme. A combination of FEM and FVM leads to the discontinuous Galerkin (DG) finite element method. This utilizes the space of basis functions in FEM but satisfies the equation similar to the FVM. This formulation gives the flexibility to ensure stability by designing a numerical flux and still can achieve high-order accuracy on general grids [1].

The generic properties of widely used numerical methods are summarized in Table 1.1. This table includes the basic properties of the methods. For further information, one can refer to [1].

These conventional methods are not suitable for problems including extremely large deformations of the mesh such as the simulation of manufacturing processes such as extrusion and molding [5]. Especially for the problems involving moving discontinuities, these methods struggle since the mesh lines should remain coincident with these regions. The most feasible solution is remeshing. However, this approach introduces a need to project the solution on a new mesh and this causes numerous difficulties. Meshless methods eliminate this problem by approximating the solution in terms of nodes. The starting point of the meshless method is the smoothed particle hydrodynamics (SPH) [6]. This method is initially proposed for simulations of astrophysical problems involving fluid masses in the absence of boundaries. This method uses kernel estimation at particles to solve the governing system of equations. The rate of

change is calculated from particle interactions. However, setting boundary conditions is not as forward as the grid-based methods and needs special treatments [7]. Nayroles et al. [8] introduced the diffuse element method (DEM). This method is based on moving least squares interpolation functions. This provides smooth approximations of function values across irregular data points [9]. The reproducing kernel particle method was introduced by Liu et al. [10]. This is a correction of the SPH method which introduces a new correction kernel for the original function of SPH [11].

1.2 Machine Learning for the Solution of Differential Equations

Besides these conventional methods, machine learning techniques are also used to solve differential equations. These regression methods offer effective and mesh-free approaches [12]. In [13], the authors use a neural minimization algorithm to solve the finite difference equations. Using this approach, they tackle the numerical load of the finite difference equations with a highly parallel algorithm. In Ref. [14], a method to solve initial and boundary value problems is presented using artificial neural networks (ANN). This method relies on the function approximation capacity of neural networks. They generalize their approach to ordinary differential equations (ODE), systems of ODEs, and PDEs. The solutions are compared with Galerkin finite element solutions, and the results match these numerical solutions.

Recently, with the growth of computing resources, advances in machine learning technology, and the availability of mass amounts of data, these techniques have increased their popularity in approximating the solution of differential equations. These low-fidelity surrogate models aim to decrease computational time with acceptable accuracy in the model. However, these surrogates cannot achieve great accuracy by themselves. This leads to multifidelity models, which were first introduced in a linear, auto-regressive, Bayesian manner by Kennedy & O'Hagan [15]. They introduced the linear relation between the low and high-fidelity models using data-driven Gaussian Processes regression [16]. This technique requires a kernel function to define the prior covariance between any two function values [17], and the choice of this prior highly affects the solution. This work of Kennedy & O'Hagan attracted interest in the computational science community to improve the shortcomings of the model and

to implement the modeling of stochastic dynamical systems. Perdikaris et al. [18], used this approach with Gaussian-Markov random fields to provide an accurate uncertainty quantification of Burgers equation from a random initial state with a decrease of computational time. In [19], they extended the recursive approach and introduced a multivariate recursive Gaussian processes regression to predict vector-valued random fields. They implemented this method to the stochastic Burgers equation and the stochastic Oberbeck-Boussinesq equations and got a very small absolute error, predictor mean, and standard deviation compared to the benchmark solutions. Moreover, Raissi [20] proposed a probabilistic regression framework for solving linear integro-differential equations from noisy data. Their approach returns solutions of the fields with uncertainty quantification for spatio-temporal and high dimensional problems. However, this approach lacks the solution of nonlinear operators. Perdikaris et al. [21] proposed a framework considered as a generalization of the work of Kennedy & O'Hagan to nonlinear autoregressive schemes to predict the multi-fidelity approximation of the Branin function and, for a practical application, modeling of mixed convection. Their nonlinear approach has higher predictive accuracy even with less training data compared to the classical linear, autoregressive approach. Lee et al. [22] improved the NARGP algorithm to capture the discontinuities. Since a smooth, stationary kernel cannot capture the discontinuities, they used non-stationary kernels with space-dependent hyperparameters to overcome this problem. However, this approach still has drawbacks in the form of the curse of dimensionality and overfitting. Constructing data-informed kernels, as in [23] can be a solution. They use an entirely data-driven method for forecasting the weekly average sea temperature. They use kernel methods and propose to learn that kernel from data using a cross-validation technique called Kernel Flows [24]. This multifidelity approach is also implemented with deep neural networks. Raissi and Karniadakis [25] offered a surrogate model that captures discontinuities with multi-fidelity data. The discontinuity is captured with neural networks but as a drawback, it cannot model the uncertainty, unlike the Gaussian processes. Moreover, Meng and Karniadakis [26] created a composite neural network that learns from multi-fidelity data. There is one low-fidelity deep neural network (DNN) connected to two high-fidelity DNNs.

1.2.1 Physics-Informed Neural Networks

The flexibility of defining a prior of Gaussian processes has some limitations. The Bayesian nature requires specific prior assumptions for the model and limits the prediction capability for the nonlinear problems [27]. Additionally, the need for high-fidelity data for the training phase is still a computational burden. To overcome these problems, Raissi et al. [27] offered physics-informed neural networks (PINN). A sample figure of a PINN is shown in Figure 1.1.

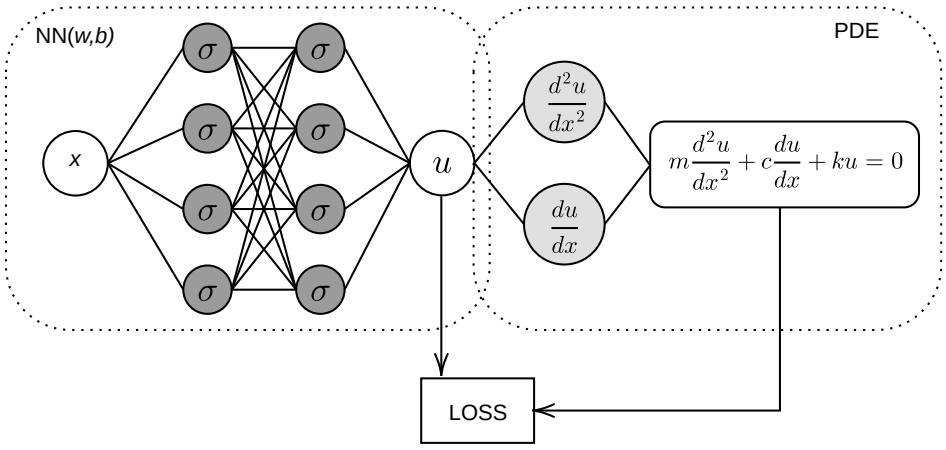


Figure 1.1: Representation of a physics-informed neural network for a 1D harmonic oscillator.

These networks use the well-known capability of the networks being universal function approximators [28] and directly solve the nonlinear problems without any prior assumptions. They use automatic differentiation [29] to differentiate the neural network prediction, therefore they can construct a PDE. These networks are used to get a solution or learn an operator in a partial differential equation. The main advantages of PINN are it does not need to generate complex meshes or use any discretization scheme. Instead, it uses point clouds generated from a statistical distribution. The discretization of a differential equation is replaced by automatic differentiation which uses consecutive chain rules to get the derivative. A sample solution of a PINN can be seen in Figure 1.2 for a 1D harmonic oscillator. A classical neural network can fit a function where data is available. However, the prediction of the area without any labeled data is completely wrong. PINNs can use the information from the differential

equation and predicts the solution well for this simple problem.

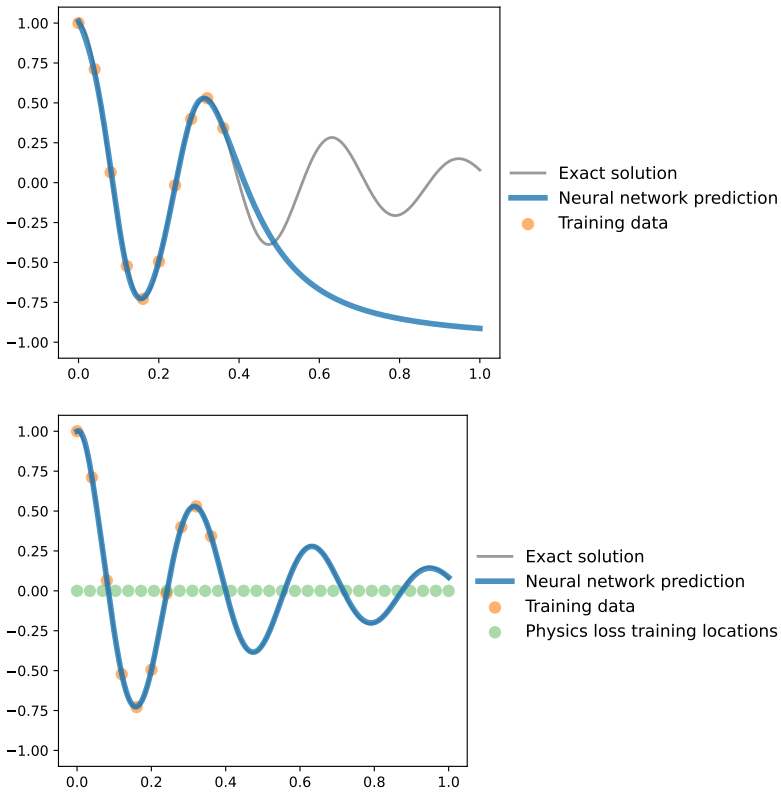


Figure 1.2: Solution of a 1D harmonic oscillator with a fully connected neural network and a physics-informed neural network. The neural network can fit a function where data is available. The PINN approach uses the differential equation and find a solution even if the regions where observations are not available.

The original paper introduced the solution of Schrödinger and Allen-Cahn equations with fully connected networks. The solution can be found by using an optimization algorithm and minimizing the loss on the points generated. The common optimization algorithm used in PINN is the stochastic gradient descent algorithm. It finds the gradient of the objective function and updates parameters in the opposite direction of the gradient. In the same paper, the authors try to find unknown differential operators with some known data points. Using the same formulation parameters in the Navier-Stokes and Korteweg-de Vries equations can be learned. With the known observations in the computational domain, one can learn the operators or even boundary conditions [30].

1.2.1.1 Application Highlights

The PINN approach got attention and has been studied widely for the solution and identification of different partial differential equations. In [31], the Navier-Stokes equations are solved with velocity-pressure and velocity-velocity formulations for the incompressible flows. Using this formulation two-dimensional Kovasznay flow, two-dimensional cylinder wake, and three-dimensional Beltrami flow are solved. For the inverse flow problems, hidden fluid mechanics is proposed [32]. The flow properties and patterns described by partial differential equations are extracted from the flow visualizations. Their algorithm is free from the geometry or the boundary/initial conditions which makes this approach very flexible in choosing the domain of interest for data acquisition. In that work, the authors extract the quantitative hemodynamics data from three-dimensional intracranial aneurysm visualization only. Solving the fluid flow problems using PINN is extended to high-speed aerodynamic flows using inviscid Euler equations [33]. A one-dimensional transient flow is used to solve a forward problem and track the position of a shock. A standard PINN approach can capture the discontinuity position. However, if the residual point generation is clustered near the discontinuity, the sharp gradient is captured more accurately. Moreover, the Boltzmann equation with the Bhatnagar-Gross-Krook (BGK) collision model is studied with PINN to solve forward and inverse problems of multiscale flows [34]. This is a well-known model describing flows in different regimes from hydrodynamic limit to free molecular flows. This approach is especially efficient in solving inverse problems but not for forward problems specifically for flows with low Knudsen numbers. Cai et al. added the energy equations coupled with the flow equations and solved different heat transfer problems [30]. They obtained the velocity and temperature fields for the forced and mixed convection problems with unknown thermal boundary conditions with the help of sparse temperature measurements. Moreover, they presented solutions to power electronics problems of realistic industrial applications. This shows the capability of PINN to tackle problems at the industrial level. This paper also includes a solution to a Stefan problem, but this is solely examined by Wang and Perdikaris [35]. They proposed a multi-network approach for the moving free boundary problems. The moving boundary is also an unknown in free boundary or Stefan flows. They parametrized the free boundary by a neural network, different

than the network constructed to find the latent solution. These network outputs are then coupled with the Stefan condition to find the solution.

1.2.1.2 Improvements on Convergence and Accuracy

The physics-informed neural networks can converge to a solution for simple problems. Despite this, PINNs can have difficulties with the problems having multiphysics or multiscale behavior [12]. The loss function of PINN generally consists of multiple terms, including residual and boundary condition losses. This yields highly non-convex optimization problems [36]. Several different approaches are presented to tackle different weaknesses of PINN. Jagtap et al. [37] proposed conservative PINN (cPINN). This approach is a domain decomposition technique for solving conservation laws. They partitioned the domain into numerous subregions. This approach offers the flexibility to apply different neural networks for each region. Prior to the knowledge of physics in each region, the neural network can be tailored to obtain more accurate solutions. Between the region interfaces, the flux continuity in the strong form is enforced to imply the conservation property. Since this approach uses domain decomposition, the algorithm can be implemented in parallel. This proposed cPINN formulation is extended by Jagtap and Karniadakis [38] named as extended physics-informed neural networks (XPINN). This approach works on any type of PDE by applying the interface conditions according to the problem itself such as solution continuity, flux continuity, etc. The major addition from cPINN is that it can be employed for any arbitrary complex domains, especially in higher dimensions. This also gives the flexibility to decompose the domain in any arbitrary way, increasing the representation and parallelization capacity. . Domain decomposition in PINN is used in [39] as *hp*-VPINN. These are variational physics-informed neural networks with *hp* refinement. The paper focuses on variational formulation based on the Petrov-Galerkin subdomain method with the trial space as the space of neural networks, and the test space is high-order polynomials. The domain decomposition allows h refinement and the projection onto space of high-order polynomials allows p refinement to the offered approach. This formulation works efficiently, especially when there exists singularities or sharp changes in the solution.

The proposed formulations above are attempts to improve the general PINN formulation and solve specific problems that the original formulation has. However, due to the nature of the loss function in PINN, the convergence of a solution is not guaranteed. The different terms in the loss function can affect different amounts for the overall convergence of the model. There are several methodologies proposed PINNs to converge to desired functions, and accelerate the convergence. Jagtap et al. [40] employed adaptive activation functions for physics-informed neural networks to accelerate the convergence rate. This adaptive term is introduced in the activation function as a scalable hyper-parameter and trained throughout the training process. By solving various problems, authors show that this approach not only improves the convergence but increases the accuracy also. Similar to this approach, the authors in [41] introduced layer-wise and neuron-wise adaptive activation functions for PINN. They additionally added a slope recovery term in the loss function. Although these additions increase the computational cost, they increase the accuracy and convergence rate of the neural networks. Moreover, some research focuses on sampling strategies to minimize the failure of convergence. In [42], the authors used an adaptive sampling strategy to solve phase field models of Allen-Cahn and Cahn-Hilliard type equations. Instead of fixed sample points, an adaptive resampling in every iteration is proposed to improve the accuracy. First, they train a network with points sampled from a specified distribution. Then, the proposed method picks a portion of points where the total error is significantly high and adds this set to the previous points, and trains the network again. This approach is particularly important for problems with moving interfaces. A similar approach is used in [43] to minimize propagation failures in PINN. The PINN solutions propagate from the proper initial/boundary conditions to the interior points. With poor sampling strategies, a solution can get stuck on a trivial solution. To prevent this, they proposed an evolutionary sampling algorithm that collects the collocation points in the regions of high PDE residuals.

In the context of achieving greater accuracy, there are some proposed formulations for the learning cycle of PINN. In [44], Wang et al. analyzed the neural networks' gradient pathologies and found a mode of failure related to numerical stiffness. This failure mode leads to unbalanced gradients during the training of the PINN model. To overcome this problem, the authors offer a learning rate annealing algorithm. This al-

gorithm identifies the unbalanced gradients, and adaptively assigns different weights to different loss terms. In another approach, Wang et al. [45] analyzed the training dynamics of PINN in the context of neural tangent kernel (NTK) theory [46]. With this formulation, it is stated that fully connected PINNs converge to Gaussian processes at the infinite width limit. Additionally, the NTK of PINN converges to a deterministic kernel under suitable assumptions. This kernel remains constant during the training process with an infinitesimally small learning rate. This shows that fully connected PINNs suffer from spectral bias [47] that prevents the networks from learning high-frequency functions. Analyzing PINNs in the context of NTK allows them to observe the training dynamics. With this observation, they observed the discrepancy of different terms in the loss function on the convergence rate. To address this, they proposed an algorithm that uses the eigenvalues of the NTK of PINN and adaptively changes the coefficients of different loss terms. This balances the convergence rate of different terms in the overall training error.

1.3 Motivation and Contributions

Physics-informed neural networks offer a promising approach for solving different types of PDEs without needing complex mesh generation and discretization schemes. Using an optimization algorithm to minimize a PDE residual, they can tackle a wide range of forward and inverse problems. Especially, in the context of tasks that do not need a high degree of accuracy, PINNs can give insight into the problem in a quick manner. The code development process of PINN is very fast compared to the traditional numerical methods. PINN does not need any discretization scheme and therefore the construction of a PDE consists of a couple of lines of code. This is one of the main motivations to use PINN for solving PDEs.

The aim of this research is to test physics-informed neural networks for computational fluid dynamics problems. This includes the solution of simple problems as well as the indirect formulations that help to get a solution. In the context of these requirements, this research compromises the following,

- Physics-informed neural networks are analyzed to get a solution for incom-

pressible and compressible flow equations. Different sampling strategies are analyzed for low Reynolds number incompressible problems to test the PINN framework. The compressible flow problems show the capability of PINN to capture discontinuity.

- Solutions for two-dimensional thermal convection with Boussinesq approximation are presented. Effects of weights in the loss function and different types of neural networks for these types of problems are analyzed. This is the first application of PINN for the solution of thermal convection problems in the literature.

- PINNs are used to get a solution for mesh deformation problems. The moving boundaries in CFD solutions require moving the mesh. With PINN, this problem can be solved with acceptable accuracy. Exact boundary enforcement is employed to ensure the boundaries are in their exact position.

1.4 The Outline of the Thesis

The remainder of the thesis study is structured as follows,

Chapter 2. The details of artificial neural networks and PINN are presented. Starting from the learning dynamics of general artificial neural networks, PINN methodology is introduced by the addition of physics loss on ANN formulation. The common activation functions and sampling strategies used in this research are presented.

Chapter 3. This chapter involves the solution of incompressible and compressible flow equations. The solution of incompressible Navier-Stokes equations for low Reynolds number flows is presented. This is extended with different PINN formulations to improve the solution. This chapter also includes the solution of compressible Euler equations with a discontinuity.

Chapter 4. Physics-informed neural networks for the solution of two-dimensional thermal convection equations are introduced in this chapter. The effect of different weight coefficients is presented for different convection regimes. Instead of fully connected networks, different types of networks are used and their difference is pre-

sented.

Chapter 5. A solution to mesh deformation problems is proposed. The mesh deformation for moving boundary problems is suitable for PINN formulation. A linear elastic method is used with hard boundary condition enforcement. This ensures the boundary is in its exact position.

CHAPTER 2

METHODOLOGY

In this chapter, the methodology of physics-informed neural networks will be discussed. First, neural networks will be considered. The learning parameters and their effect on convergence and accuracy are discussed. Then, the theory of PINN is presented.

2.1 Artificial Neural Networks

An artificial neural network (ANN) is a collection of connected artificial neurons. These neurons receive and pass information to and from other neurons. Each unit receives information from the previous units with a corresponding weight w_i. All the information from these units is summed. Then a bias is added to this sum. Finally, a function takes this sum and provides an output y. This process can be written as:

$$y = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right), \tag{2.1}$$

where σ is the activation function, and N is the number of previous information units.

The simplest neural network, generally called a fully connected network or multilayer perceptron (MLP), consists of multiple artificial units connected to each other. This type of network consists of an input and an output layer, and in between, there are hidden layers with specified depth and width and a nonlinear activation function. The schematic of a fully connected network can be seen in Figure 2.1.

A nonlinear activation function is a key to increasing the complexity of the neural network. Without such a function, a neural network is only capable of linear mapping. Some common activation functions include rectified linear unit (ReLU) [48],

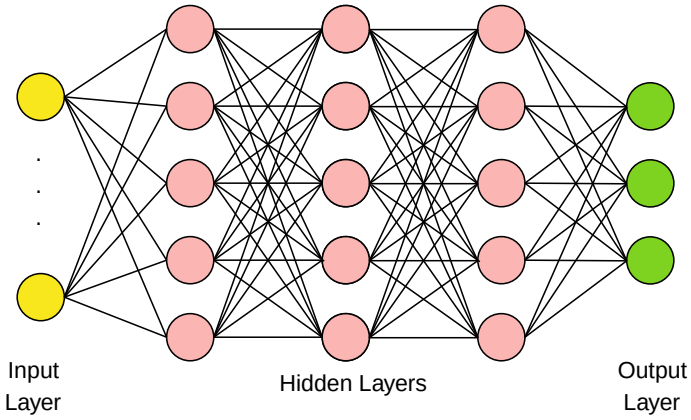


Figure 2.1: Schematic of a multilayer perceptron

hyperbolic tangent, sigmoid [49] and swish [50]. The plots of these functions can be seen in Figure 2.2.

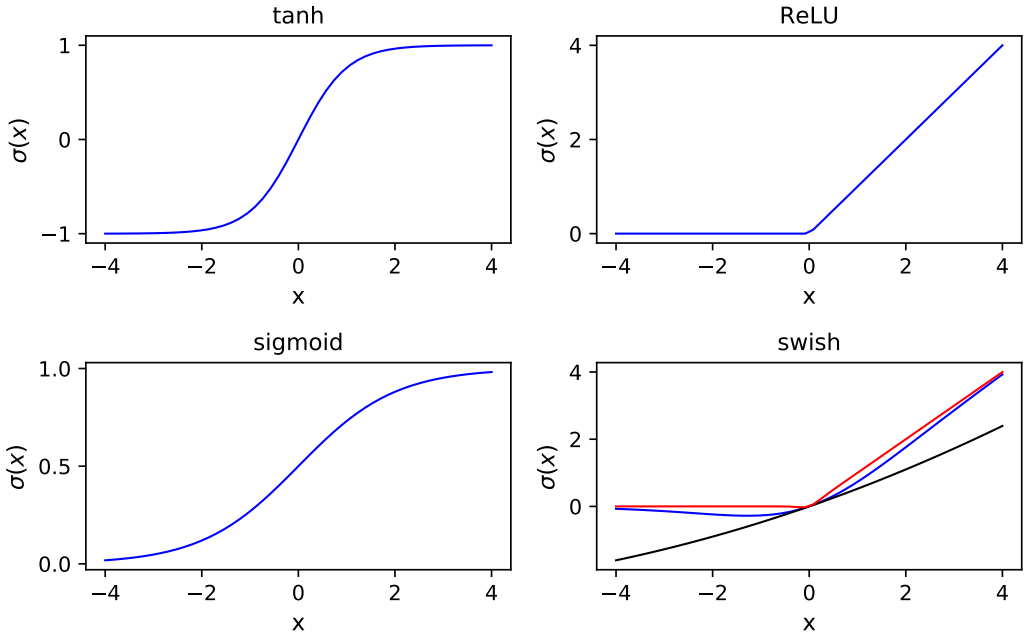


Figure 2.2: Plots of commonly used activation functions

Hidden layers in an MLP take inputs from the previous layer. Each input is multiplied by a specific weight. The nodes in the hidden layers and the output layer also have a bias term. As mentioned previously, the hidden layers have a nonlinear activation function. However, in the output layer, the activation function is an identity function

since the variable in the output layer is generally a real-valued target in regression or a value representing class scores in classification. In general, an L-layered network can be written as:

$$\mathcal{N}^0(\mathbf{x}) = \mathbf{x} \tag{2.2a}$$

$$\mathcal{N}^i(\mathbf{x}) = \sigma(\mathbf{W}^i\mathcal{N}^{i-1}(\mathbf{x}) + \mathbf{b}^i) \tag{2.2b}$$

$$\mathcal{N}^L(\mathbf{x}) = \mathbf{W}^L\mathcal{N}^{L-1}(\mathbf{x}) + \mathbf{b}^L, \tag{2.2c}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and the bias vector, respectively. $\mathbf{W}$ is an $(m \times n)$ matrix where n is the length of the input vector and m is the length of the output vector. $\mathbf{b}$ is an $(m \times 1)$ vector. This product produces an output vector y which has a dimension of $(m \times 1)$. In matrix form, computation between layers can be written as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & \dots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right) \tag{2.3}$$

Weights and biases are trainable parameters named hyperparameters, labeled with θ. The learning procedure of the neural networks is to find an optimized set of hyperparameters that minimizes a specific loss function. The loss function (or the objective function) is an indicator of how the neural network fits the model. It measures how much the output layer diverges from the desired target. One of the most common loss functions is the mean squared error (MSE), which measures the average of the square of the difference between the predicted and the true value. It can be expressed as

$$\mathcal{L}_{MSE} = \frac{1}{N_d}\sum_{i=1}^{N_d} |\hat{y}_i - y_i|^2, \tag{2.4}$$

where $\hat{y}_i$ is the neural network's output, and y_i is the target value. Before any learning procedure starts, the hyperparameters need to be initialized. Learning procedure with random initialization is doing poorly in deep neural networks [51]. To handle this issue Xavier initialization [51] is utilized. The weights are randomly sampled from a truncated normal distribution with a standard deviation of:

$$\sigma_{Xavier} = \sqrt{\frac{2}{n+m}}, \tag{2.5}$$

where n and m are the input and the output dimensions, respectively.

Since the main goal here is to predict the exact value, the loss needs to be minimized. This optimization problem can be represented as:

$$\theta^* = \arg\min_{\theta} J(\theta; \mathbf{x}), \tag{2.6}$$

where J is the objective function, such as in Equation 2.4. The hyperparameters can be updated in successive iterations by approximating the solutions to this minimization problem. This can be achieved by using one of the stochastic gradient descent (SGD) algorithms [52]. Gradient descent minimizes the objective function $J(\theta)$, by finding new parameters θ in the opposite direction of the gradient of the loss function $\nabla_\theta J(\theta)$. The gradient of this loss function with respect to hyperparameters is calculated by backpropagation [53]. It first calculates the loss by using existing weights and biases (forward pass), then finds the gradient of loss by using the chain rule while moving backward in the network. Plain gradient descent, also called batch gradient descent, calculates the gradient of the objective function for the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; \mathbf{x}), \tag{2.7}$$

where η is the learning rate. It is a parameter that determines the size of the step to take in the direction of the negative gradient to reach a local minimum [52]. The choice of the learning rate affects the behavior, and the path of the gradient descent [54]. The widely used technique for learning rate implementation is learning rate decay. This approach starts with a large learning rate at the first iterations (epochs), then reduce the learning rate gradually with the form:

$$\eta(t) = \eta_0 d^{t/T}, \tag{2.8}$$

where t is the iteration number, T is the number of steps to decay in and d is the decay rate. The effect of decaying is to start with an initially large rate to avoid memorization of noisy data and decay to a smaller value to prevent oscillations around a local minimum [55]. Batch gradient descent calculates the gradient of the whole dataset for every update. Therefore, it can be slow or can fill the memory for large datasets, but it is guaranteed to converge to the global minimum for convex optimization problems [52]. On the other hand, stochastic gradient descent performs a parameter update at

each training example of x^i:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; \mathbf{x^i}). \tag{2.9}$$

Since the parameter update is one update at a time, SGD is usually faster.

In this work, we use one of the most common optimizers named adaptive moment estimation, Adam [56]. It is an algorithm for first-order gradient-based optimization of stochastic loss functions. Most popular machine learning frameworks such as TensorFlow [57], and PyTorch [58] contains this method. One iteration of the Adam algorithm starts with the calculation of the gradient based on the previous iteration ($(t-1)$), g_t:

$$g_t = \nabla_\theta J(\theta^{(t-1)}). \tag{2.10}$$

Next, the biased first (m_t) and second raw moment (v_t) estimates are updated:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) g_t \tag{2.11a}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) g_t^2. \tag{2.11b}$$

Then, the biases are corrected for the first and second raw moment estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{2.12a}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{2.12b}$$

Lastly, the parameters are updated:

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\epsilon + \sqrt{\hat{v}_t}}. \tag{2.13}$$

β_1 and β_2 are the hyperparameters that control the exponential rate decay of the moving average of gradient m_t and the squared gradient v_t. ϵ is selected as a very small number to prevent division by zero.

2.2 Physics-Informed Neural Networks

Physics-informed neural networks (PINN), introduced by Raissi et al. [27], are neural networks trained by considering given laws of physics described by general nonlinear partial differential equations. It employs the capability of neural networks as universal function approximators [28, 49]. By utilizing automatic differentiation [29],

PINN can differentiate the neural network with respect to input coordinates. With this property, one can add the differential equation residual into the loss function. In PINN there is no need to generate a mesh which can be complex and needs expertise. Instead, a point cloud is generated in the domain and the boundaries. Automatic differentiation can use these coordinates and calculates the derivatives needed. In Figure 2.3, the schematic of a PINN is shown with the viscous Burgers' equation.

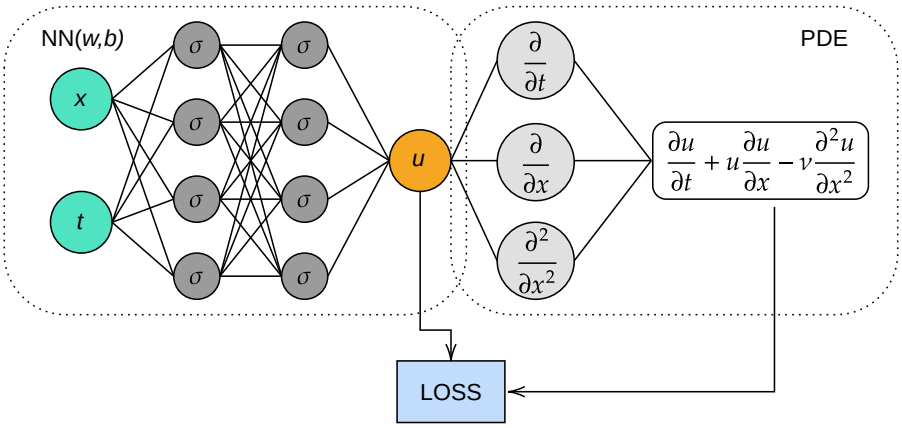


Figure 2.3: Schematic of a physics-informed neural network. The residual loss is shown with the viscous Burgers' equation.

PINN can be used in forward or inverse problems. In forward problems, it is used to approximate the physics in a domain considering the differential equation with only boundary and/or initial condition data provided. For the inverse problems, PINN can learn nonlinear continuous operators from a relatively small dataset [59]. To show the methodology of PINN, consider a general partial differential equation:

$$u_t + \mathcal{N}[u] = 0, \quad x \in \Omega,\ t \in [0,T] \tag{2.14a}$$

$$u(x,t) = g(x,t), \quad x \in \partial\Omega,\ t \in [0,T] \tag{2.14b}$$

$$u(x,0) = h(x), \quad x \in \Omega \tag{2.14c}$$

where $u(x,t)$ represents the hidden solution, $\mathcal{N}[\cdot]$ is a generalized nonlinear operator, $g(x,t)$ represents the boundary conditions and $h(x)$ is the initial condition. The hidden solution can be approximated by a feedforward neural network, $\hat{u}(x,t)$. However, for PINN the objective function is a composite loss function in the form:

$$\mathcal{L} = w_R\mathcal{L}_R + w_{BC}\mathcal{L}_{BC} + w_{IC}\mathcal{L}_{IC}, \tag{2.15}$$

where w terms are specific weighting on the total loss. Using mean squared error for the loss function, the terms become:

$$\mathcal{L}_R = \frac{1}{N_R}\sum_{i=1}^{N_R} |\hat{u}_t + \mathcal{N}[\hat{u}]|^2 \tag{2.16a}$$

$$\mathcal{L}_{BC} = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}} |\hat{u}(x^i, t^i) - g(x^i, t^i)|^2 \tag{2.16b}$$

$$\mathcal{L}_{IC} = \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}} |\hat{u}(x^i, 0) - h(x^i)|^2 \tag{2.16c}$$

$\mathcal{L}_R$, $\mathcal{L}_{BC}$, and $\mathcal{L}_{IC}$ represent the residual of the governing PDE, the boundary conditions, and the initial condition respectively. N_R, N_{BC}, and N_{IC} are the number of sampled data points for different terms. Using automatic differentiation to calculate the derivatives with respect to input coordinates, this formulation forms the total loss with the residual, boundary, and initial conditions. Optimization algorithms mentioned in the previous section minimize this objective function. This leads to predicting a field that represents the governing physical laws.

2.2.1 Activation Functions in PINN

Consider a Poisson equation as the model problem.

$$u_{xx}(x) = f(x), \quad x \in \Omega \tag{2.17a}$$

$$u(x) = g(x), \quad x \in \partial\Omega. \tag{2.17b}$$

If one wants to predict a solution with a PINN consisting of only one hidden layer. We can write the network explicitly as:

$$u(x;\theta) = \mathbf{W}^1 \cdot \sigma(\mathbf{W}^0 x + \mathbf{b}^0) + \mathbf{b}^1, \tag{2.18}$$

where $\theta = (\mathbf{W}^0, \mathbf{W}^1, \mathbf{b}^0, \mathbf{b}^1)$ represents all the parameters in the network. Then, the second derivative is straightforward such that,

$$u_{xx}(x;\theta) = \mathbf{W}^1 \cdot \left[\ddot{\sigma}(\mathbf{W}^0 x + \mathbf{b}^0) \odot \mathbf{W}^0 \odot \mathbf{W}^0\right], \tag{2.19}$$

where $\odot$ represents element-wise multiplication. The term $\ddot{\sigma}$ shows there is a need to select an activation function that is two times differentiable. The second-order

derivative is present for most of the engineering applications of differential equations. Therefore, in PINN, the choice of activation function is generally a two times differentiable function. The activation function plays an important role in training PINNs. However, there is no obvious choice for it since it is problem-dependent [40]. To tackle this issue, adaptive activation functions are presented in the literature [40]. In this approach, there is a learnable parameter inside the activation function such that,

$$\sigma(\alpha(Wx + b)). \tag{2.20}$$

This new term α is also a parameter that needs to be trained and optimized by SGD algorithms. In every learning epoch the parameter update has an additional term of

$$\alpha = \alpha - \eta \cdot \nabla_{\alpha} J(\alpha). \tag{2.21}$$

The authors in [40] use this adaptive approach for solving the sine-Gordon equation, Helmholtz equation, Klein-Gordon equation, and Burgers equation. This trainable hyperparameter in the activation function increases the convergence of the neural network and also the accuracy compared to fixed activation functions.

2.2.2 Sampling of Points

One of the important factors in PINN is to sample the collocation points, which are used to calculate the residual of the governing PDE. Residual minimization in these interior points in the domain Ω, can stuck at the trivial solution $u(x, t) = 0$ for any homogeneous PDE, if the provided data for the correct initial/boundary condition is not sufficient. Also, to minimize the loss, PINN requires the gradient of the output with respect to input data. Therefore, the solution at an interior point is affected by the solutions at nearby points. In order for PINN to converge to the correct solution, it must propagate from the known initial/boundary points to the collocation points [43]. The basic method is to sample all the points before starting the learning procedure. This can be done by sampling the points from a statistical distribution. Common methods are sampling randomly from a uniform distribution or Latin hypercube sampling [60]. However, to prevent propagation failure, different sampling strategies are present. Although there are numerous strategies, such as evolutionary sampling [43] and importance sampling [61], the dynamic random sampling strategy and fixed set

of sampled points are used in this thesis. It is a simple strategy that dynamically samples a random set of collocation points at every iteration. It is computationally cheaper than the method that samples all the points at once since, generally, the set in dynamic sampling is smaller.

CHAPTER 3

PINN FOR INCOMPRESSIBLE & COMPRESSIBLE FLOW EQUATIONS

3.1 Incompressible Navier-Stokes Equations

Most flows generated by nature or human industrial applications can be modeled with Navier-Stokes (NS) equations. Therefore these equations are widely studied in engineering research. Physics-informed neural networks are also used to solve the Navier-Stokes equations in the literature. These simulations are restricted to low Reynolds number flows since PINN can have difficulties in learning the solutions of multiscale and time-dependent problems. Resolving turbulence needs the addition of data [62] or fully developed turbulent initial conditions [32]. In this section, we will analyze a classical benchmark case at a low Reynolds number to test the PINN framework for incompressible Navier-Stokes equations. All experiments are conducted on a Dell Precision mobile workstation with an NVIDIA RTX A1000 laptop GPU and an Intel Core i7-12800H processor.

3.1.1 Formulation

The steady, incompressible, laminar flow of fluids is governed by the Navier-Stokes equations defined on the domain Ω. Domain boundary is represented with $\partial\Omega$. The non-dimensional incompressible Navier-Stokes equations and the continuity equation on the domain Ω can be written as

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\Delta\mathbf{u}, \quad \mathbf{x} \in \Omega, \tag{3.1a}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{3.1b}$$

where $\mathbf{u}$ and p are the non-dimensional velocity and the scalar pressure field, respectively. The non-dimensional quantities are obtained with the following parameters.

$$x = \frac{x^*}{L_r},\ u = \frac{u^*}{U_r},\ p = \frac{p^*}{\rho_r U_r^2},\ \rho = \frac{\rho^*}{\rho_r},\ \nu = \frac{\nu^*}{\nu_r}. \tag{3.2}$$

Here, the superscript * denotes the dimensional parameters, and the subscript r denotes the reference values. The non-dimensional Reynolds number is defined as $Re = U_r L_r / \nu_r$.

3.1.2 Results

The lid-driven cavity flow is a classical benchmark problem widely studied to test the numerical methods for incompressible Navier-Stokes equations [2, 63, 64]. The computational domain is $\Omega = [0, 0] \times [1, 1]$, boundaries are no-slip walls except the boundary at $y = 1$, where there is a horizontal lid velocity taken as $U_{lid} = 1$.

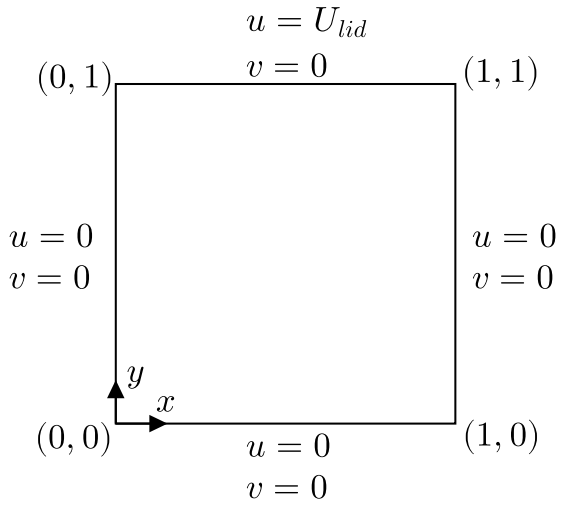


Figure 3.1: The schematic and the boundary conditions of the lid-driven cavity benchmark problem.

First, a grid search methodology is conducted to find the optimal width and depth of the network. A series of simulations are performed with 50000 iterations and a learning rate of 5×10^{-3} for Reynolds number $Re = 100$. A dynamic sampling

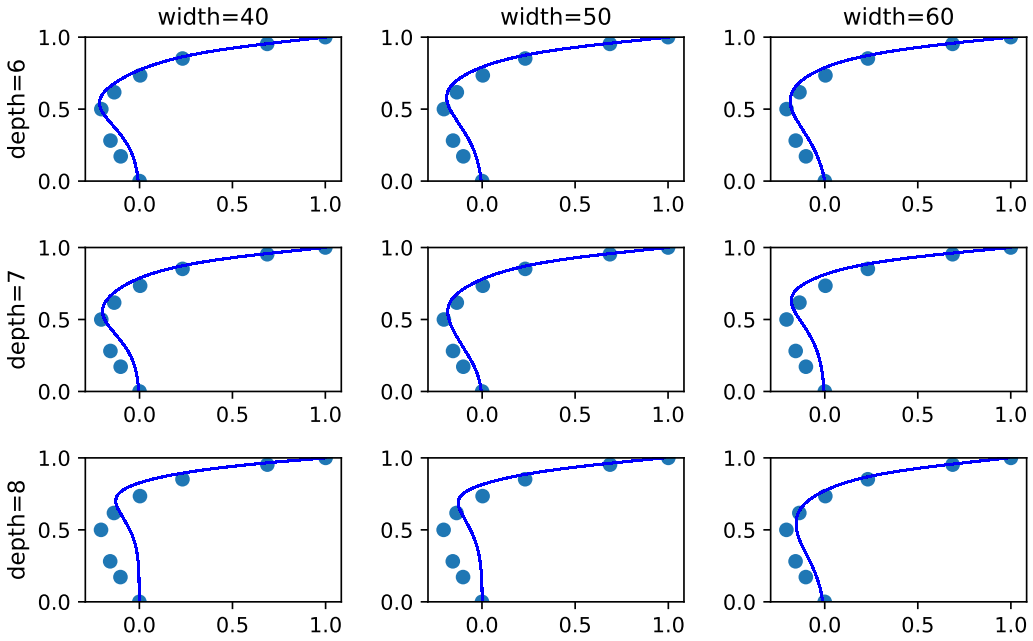
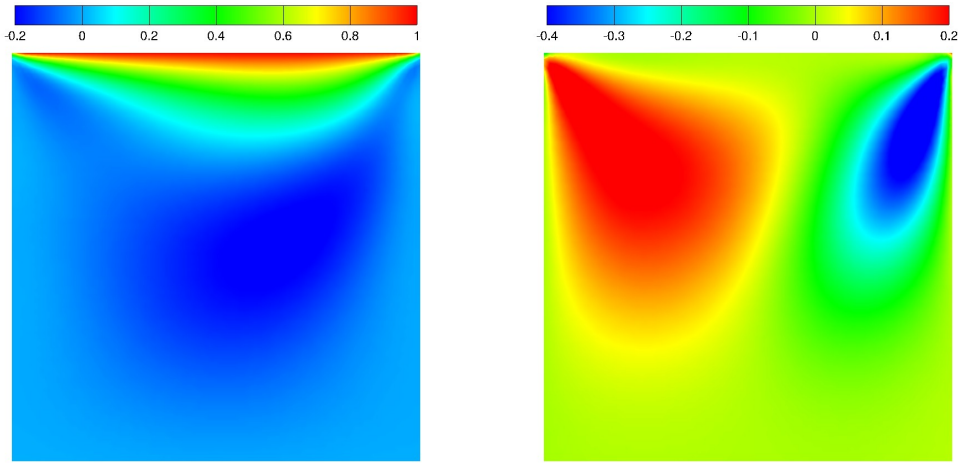


Figure 3.2: Results compared with Ghia's reference solution [2] for the grid search study. The u velocity at $y = 0.5$ is plotted. The scattered data is the reference solution and the plotted curve is the PINN solution. The horizontal axis represents u velocity and the vertical axis represents the y coordinate.

strategy [43] is used for all simulations. This strategy samples a different number of points from a random uniform distribution for every iteration of the optimization process. A batch size of 128 is used for the collocation points and each boundary. The u velocity is plotted at $y = 0.5$ line, and the results are compared with Ghia's reference solution [2]. The comparison can be seen in Figure 3.2.

In most of the results, the trend of the u velocity of the PINN prediction is similar to the reference solution. For the rest of the study, the solution with 7×50 is used since it represents the most similar result.

The velocity field with the 7×50 configuration is presented in Figure 3.3. The u velocity field is presented on the left, and the v velocity field is on the right. The PINN solution captures the trend of the flow well. However, near the corners of the top boundary, the solution deviates from the common trend. The lid velocity is not in its true value of 1, and the v velocity cannot propagate through the corners.



(a) (b)

Figure 3.3: Velocity field of the lid-driven cavity problem. The solution is obtained by a 7×50 PINN configuration. The figure (a) shows the u velocity profile while the figure (b) shows the v velocity profile

To show the difference in sampling strategies, a new solution is obtained with a fixed sample of points in the computational domain. This approach requires higher computational time since the number of loss calculations and automatic differentiation in every step increases. A fixed 1000 points are sampled from a uniform distribution for the computational domain. Similarly, 120 points are sampled for each boundary. The solution is obtained with the same configuration in 50000 iterations. The loss history of the training is presented in Figure 3.4. The training time for mini-batch solution is 40 minutes, while the training time for the solution with fixed sample of points is roughly 2 hours.

The loss history shows that compared to mini-batch training, using a fixed set of points results in a lower loss after 50000 iterations for this problem. Moreover, the oscillations in the overall loss get lower. This is due to the nature of mini-batch sampling. In every iteration, the sampled 128 points are in different regions. Every point contributes differently to the loss function according to the region of interest.

The loss history of each term for the fixed set of sampling is shown in Figure 3.5.

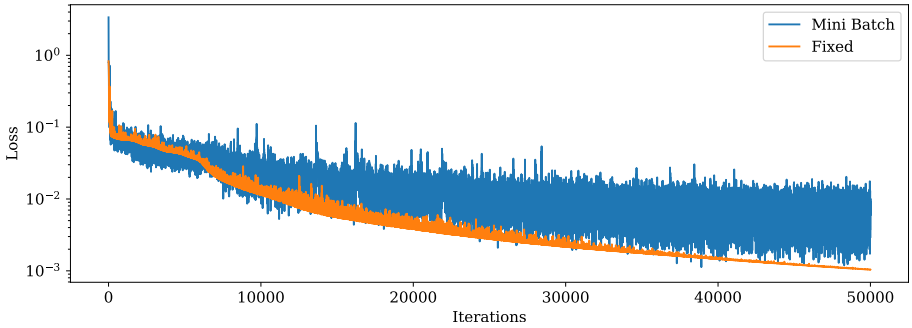


Figure 3.4: Loss history of the training of PINN. A mini-batch solution with 128 sample points in each iteration is represented along a fixed set of training with 1000 collocation points and 120 boundary points.

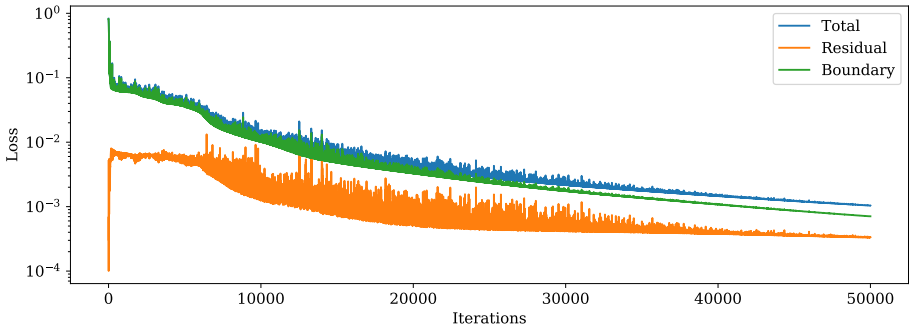


Figure 3.5: Loss history of the fixed set of points training of PINN. The history of each loss term is presented

The residual loss starts from a very low number resembling the trivial solution. It immediately jumps back to a higher loss with boundary term. The overall loss mostly comes from the supervised boundary loss. For this problem, there are discontinuities in the corners affecting the trend of the loss. The profile of the residual term and the boundary term looks similar. Although the effects of each term on the overall loss function are different, they are correlated. The residual solution is propagated from the boundary to the interior points [43].

3.2 Compressible Euler Equations

In this study, physics-informed neural networks have been applied to solve compressible Euler equations. As a pioneering study, the progression of a one-dimensional, time-dependent shock has been examined. The details of the computation domain and the artificial neural networks used for the solution are presented, and the obtained results are compared with the real solutions. The following loss term is used in PINN to solve the Euler equations.

$$\mathcal{L} = w_D\mathcal{L}_D + w_R\mathcal{L}_R + w_{BC}\mathcal{L}_{BC} + w_{IC}\mathcal{L}_{IC}. \tag{3.3}$$

The term $\mathcal{L}_D$ term indicates the loss on the observed data used in this case. Similarly, w_D is the weighting term for this loss.

In an inviscid, compressible flow, mass, momentum, and energy conservation can be modeled using the Euler equations. These equations can be written as:

$$\partial_t U + \nabla \cdot f(U) = 0,\ x \in \Omega \subset \mathbb{R}, \quad t \in [0, T] \tag{3.4}$$

These terms U and $f(U)$ are written as

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho E \end{pmatrix}, \quad f(U) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ u(\rho E + p) \end{pmatrix}$$

Here, ρ is the density, u is the one-dimensional velocity, p is the pressure and E is the total Energy. In addition to these equations, a state equation is also required. This equation is:

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho u^2\right).$$

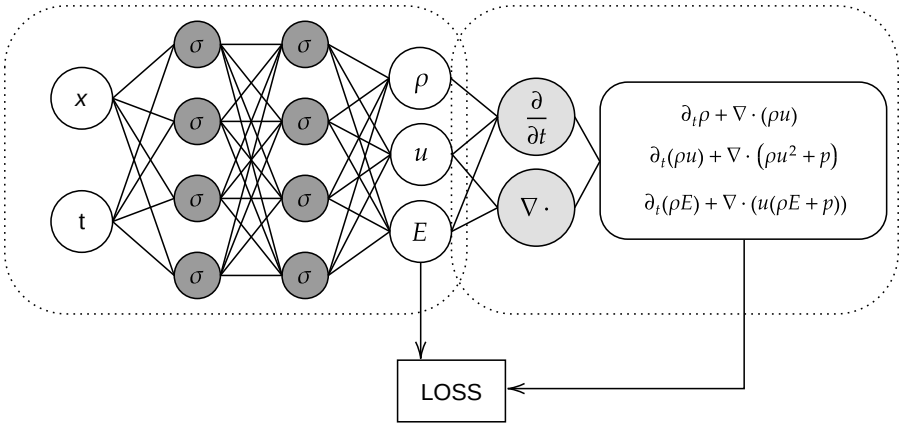


Figure 3.6: Schematic of a physics-informed neural network for the solution of compressible Euler equations.

For air as the working fluid, γ can be taken as 1.4.

As shown in Figure 3.6, the PINN take the spatial and temporal coordinates as input and passes them through hidden layers to estimate the density (ρ), velocity (u), and energy (E) terms required for the compressible Euler equations. Then, these variables are placed in the boundary conditions and initial conditions. The losses are calculated at the sampled boundary, initial, and sampled collocation points, and the derivatives are calculated with automatic differentiation. These calculations are placed in the compressible Euler equations to ensure that they are satisfied. The total error function consists of the sum of these errors and is tried to be brought close to zero with the Adam optimization algorithm.

3.2.1 Results

The following settings are used to demonstrate the solution for the compressible Euler equations. The computation domain is between $[0, 1]$ and a shock is located at the point $x = 0.5$ as the initial condition. The initial values on the left and right sides of the shock are:

$$(\rho_L, u_L, p_L) = (1.4, 0.1, 1.0), \quad (\rho_R, u_R, p_R) = (1.0, 0.1, 1.0).$$

The exact solution is used as a Dirichlet boundary condition [33]. The exact solution is

$$\rho(x,t) = \begin{cases} 1.4 & x < 0.5 + 0.1t, \\ 1.0 & x > 0.5 + 0.1t, \end{cases} \qquad u(x,t) = 0.1, \quad p(x,t) = 1.0$$

For the training phase of the artificial neural networks, 60 boundary points, 60 initial points, and 1000 points within the computation domain were randomly selected using Latin Hypercube sampling. In addition, 150 values from real solutions were also added to the artificial neural network in a different calculation. This selection is shown in Figure 3.7 and the points marked in purple are obtained by randomly selecting the real values. The solution is analyzed with a maximum time of $t = 2.0$s. In this study, time is treated as a continuous variable such as the spatial variable x. The network structure includes 4 hidden layers with 40 neurons in each hidden layer. The Adam optimizer with a learning rate of 0.001 and 20000 iterations was used to calculate the optimal parameters, followed by the BFGS (Broyden–Fletcher–Goldfarb–Shanno) method for fine-tuning. The hyperbolic tangent function is used as the activation function. The training time is roughly 5 minutes for all experiments on a Dell Precision mobile workstation with an NVIDIA RTX A1000 laptop GPU and an Intel Core i7-12800H processor.

In Figure 3.8, the density contour of the PINN solution is presented. The initial and boundary condition are well presented as we can see the initial discontinuity at the point $x = 0.5$. This initial shock moves on the spatial domain according to Equation 3.4. The discontinuous region can be seen such that the density contour changes its value in a very sharp manner. In Figure 3.9, the prediction errors of the pressure and velocity fields are shown. The errors are relatively higher near the discontinuous region.

In Figure 3.10, the comparison of PINN solutions with the exact solution at time $t = 2.0$s is shown. The PINN formulation can predict the position of discontinuity well. However, it cannot capture the sharp change and small oscillations are observed near the discontinuous region. To suppress these oscillations, we can add labeled data. The figure on the right shows the solution with additional 150 data from the exact solution. This addition helps regularize the PINN into a more accurate solution

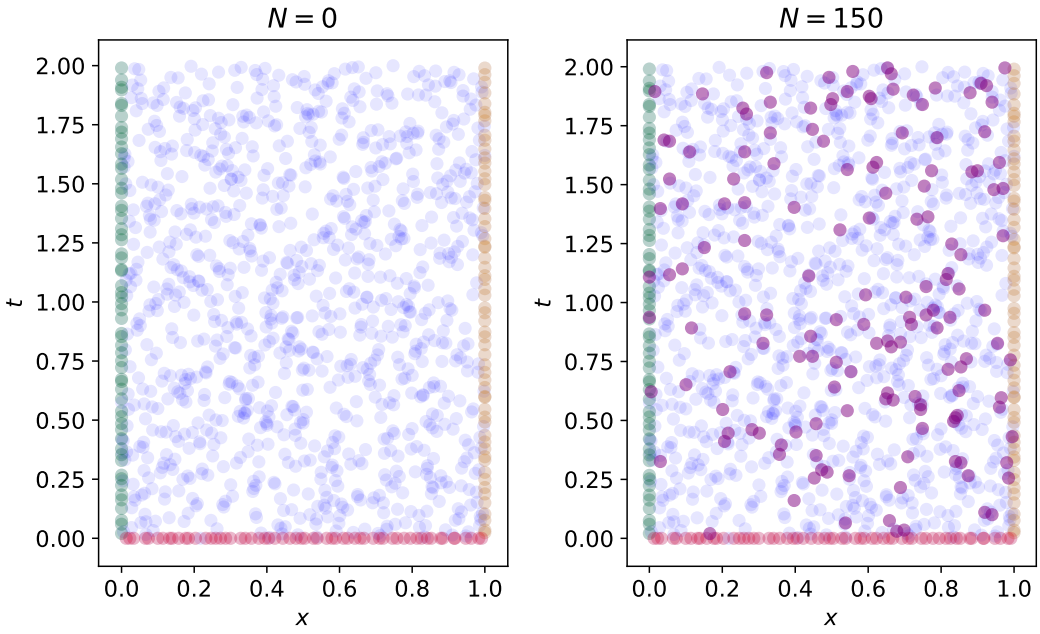


Figure 3.7: Computational domain and sampled points for the solution of compressible Euler equations with PINN. The figure on the right shows the addition of 150 exact solution points.

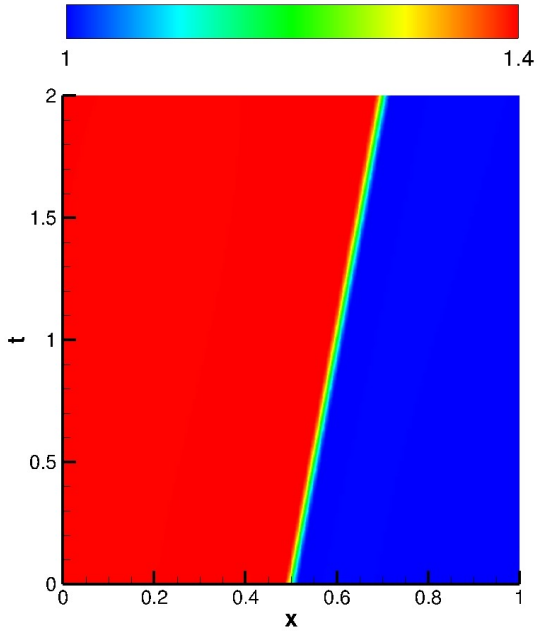
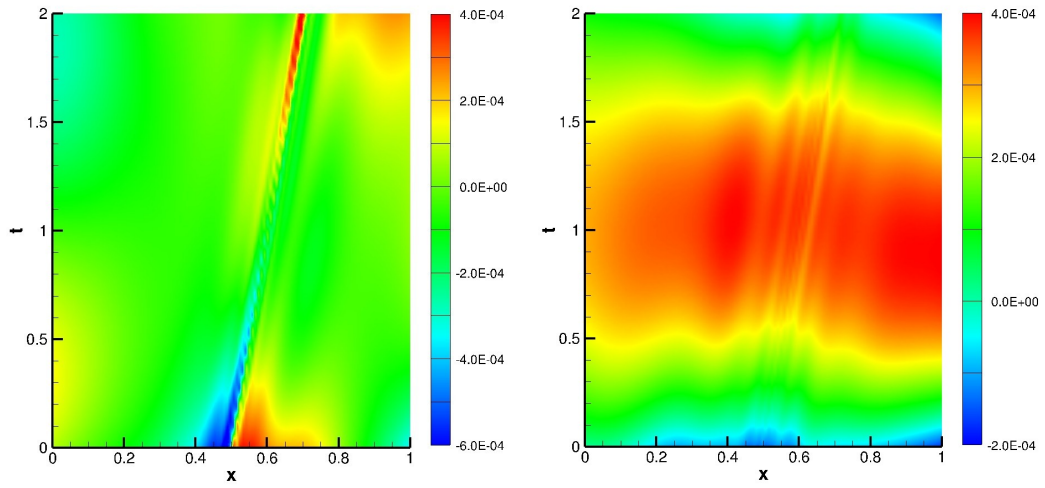


Figure 3.8: Density contour of the solution of Euler equations



(a) (b)

Figure 3.9: Error fields of the solution to Euler equations. Figure (a) shows the pressure error profile while figure (b) shows the velocity error profile without any additional true observations

and the sharp gradient can be captured.

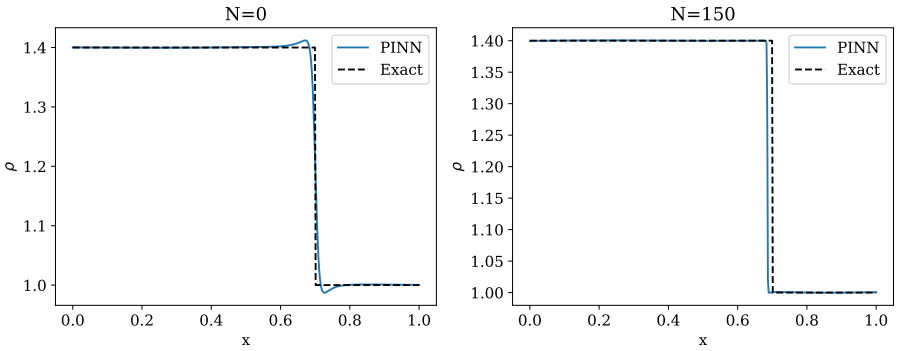


Figure 3.10: Line plot of the density at time $t = 2.0$s compared with the exact solution. The first figure shows the density values with only boundary and initial data training. Figure right is the PINN solution trained with the exact solution addition.

CHAPTER 4

PHYSICS-INFORMED NEURAL NETWORKS FOR THERMAL CONVECTION PROBLEMS

Thermal convection problems arise in many practical engineering applications, such as the cooling of electronic chips. This type of real-life analysis of fluid flow and heat transfer requires high degrees of freedom to minimize numerical error. This can be achieved with high-quality mesh or high-order discretizations. However, mesh generation is time-consuming and needs expertise, and high-order simulation tools for this type of problem are computationally demanding.

The incompressible thermal convection is studied in the literature with various numerical methods [65]. [66] used a least squares finite element method based on a velocity, pressure, vorticity, temperature, and heat flux formulation for time-dependent problems. [67] developed a spectral/hp element method for the Direct Numerical Simulation (DNS) of incompressible thermal convective flows by considering Boussinesq-type thermal body-forcing with periodic boundary conditions and enforcing a constant volumetric flow rate. In [68], the author presented a GPU-accelerated nodal discontinuous Galerkin method on unstructured triangular meshes for solving problems on different convective regimes.

Due to the popular deep learning frameworks such as TensorFlow [57], and PyTorch [58], and their easy implementation, PINNs have become quite popular for solving PDEs. Moreover, there are some software libraries specifically designed for physics-informed machine learning such as DeepXDE [69] and NeuralPDE [70]. It is used for solving incompressible and compressible Navier-Stokes equations [71, 31, 72], as well as in inverse heat transfer problems [30].

In this work, we present the application of PINNs to coupled fluid flow and heat transfer problems in different thermal convection regimes. In particular, we show the accuracy of the prediction increases by adding numerous true observations or high-fidelity data. As the number of high-fidelity data increases the accuracy of the solution increases. This model relies on the specific weights assigned to different loss terms in a composite loss function. We show that changing these specific weights can increase accuracy according to the problem where boundary conditions or flow inside the computational domain dominates. In addition, for thermal convection problems, different types of networks can be used instead of simple fully connected networks. We show how different networks can change the convergence rate of the total training error. All the experiments are conducted on a machine with Intel Core i7-6700HQ CPU and Ubuntu 20.04 operating system.

4.1 Formulation

We consider a two-dimensional domain $\Omega \subset \mathbb{R}^2$ and denote the boundary of Ω by $\partial\Omega$. Following the notation presented in [73], we denote the Dirichlet and Neumann boundary conditions as $\partial\Omega_D$ and $\partial\Omega_N$ respectively on $\partial\Omega$. We are interested in the approximation of non-isothermal incompressible Navier-Stokes equations coupled with the energy equation through the Boussinesq approximation, which reads:

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{in } \Omega \times (0, T] \tag{4.1a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\nabla p + \frac{1}{Re}\Delta \mathbf{u} + \mathbf{s}_{\mathbf{u}} \qquad \text{in } \Omega \times (0, T], \tag{4.1b}$$

$$\frac{\partial \theta}{\partial t} + (\mathbf{u} \cdot \nabla)\,\theta = \frac{1}{RePr}\Delta \theta + \mathbf{s}_{\theta} \qquad \text{in } \Omega \times (0, T], \tag{4.1c}$$

in non-dimensional form and subject to the initial conditions

$$\mathbf{u} = \mathbf{u}_0, \quad \theta = \theta_0 \quad \text{for } t = 0, \mathbf{x} \in \Omega, \tag{4.2}$$

and the boundary conditions

$$\mathbf{u} = \mathbf{g}_D \ \text{ on } \ \mathbf{x} \in \partial\Omega_D^{\mathbf{u}}, t \in (0, T], \tag{4.3a}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{n}} = 0, \ p = 0 \ \text{ on } \ \mathbf{x} \in \partial\Omega_N^{\mathbf{u}}, t \in (0, T]. \tag{4.3b}$$

$$\theta = g_D \ \text{ on } \ \mathbf{x} \in \partial\Omega_D^{\theta}, t \in (0, T], \tag{4.3c}$$

$$\frac{\partial \theta}{\partial \mathbf{n}} = g_N \ \text{ on } \ \mathbf{x} \in \partial\Omega_N^{\theta}, t \in (0, T]. \tag{4.3d}$$

Here $\mathbf{u}$, p, and θ are non-dimensional velocity, static pressure, and temperature fields, respectively. The following approach is used to get the non-dimensional representation of the equation.

$$\begin{aligned} x = \frac{x^*}{L_r}, \ t = \frac{t^*}{L_r/U_r}, \ u = \frac{u^*}{U_r}, \ p = \frac{p^*}{\rho_r U_r^2}, \\ \rho = \frac{\rho^*}{\rho_r}, \ \nu = \frac{\nu^*}{\nu_r}, \ \alpha = \frac{\alpha^*}{\alpha_r}, \ \theta = \frac{T - T_r}{T_s} \end{aligned} \tag{4.4}$$

Here the superscript * denotes the dimensional parameters and subscript r denotes the reference values for specific terms with reference length scale L_r, velocity U_r, density ρ_r, viscosity ν_r, thermal diffusivity α_r, and temperature T_r. The non-dimensional Reynolds and Prandtl numbers are defined as $Re = U_r L_r/\nu_r$ and $Pr = \nu_r/\alpha_r$. $\mathbf{s_u} = (\mathbf{g}\beta\,(T - T_r)\,L_r/U_r)\,\theta$ is the forcing term for Navier-Stokes, where $\mathbf{g}$ is the gravitational acceleration, β is the expansion coefficient and subscript r refers the reference value for the corresponding field. $s_\theta = s_\theta\,(\theta, \nabla\theta, \mathbf{u})$ is the generic generation term for the energy equation written in terms of temperature. The superscripts $\mathbf{u}$ and θ in boundary representation separate the Dirichlet and Neumann conditions, represented with the subscripts D and N, on the physical boundary set for flow and heat transfer equations.

4.2 Results

We have implemented our physics-informed neural network on top of the NVIDIA Modulus framework [74]. We use the Adam optimizer [56] to minimize the loss function of the PDE and use 8 hidden layers with 40 units for each test case where the neural network parameters are initialized using the Glorot scheme [51]. We solve different 2D thermal convection tests to show the solutions by representing the velocity, pressure, and temperature fields.

4.2.1 Poiseuille Flow

We consider a two-dimensional channel flow with a fully developed Poiseuille profile in the first test case. The channel dimension is $[0, 2] \times [-1, 1]$. The upper and lower walls have constant temperatures of $\theta_L = 1$ and $\theta_U = 0$. No-slip boundary conditions are imposed for upper and lower walls. The fully developed solution of the velocity field with linear temperature profile shown below is implemented as the boundary conditions of the inlet and the outlet. The flow conditions are stated as $Ra = 10^3$, $Pr = 0.71$ and $Re = 100$.

$$u = 1 - y^2, \quad v = 0, \quad p = \frac{Ra}{2PrRe^2}\left(y - \frac{y^2}{2}\right) - \frac{2x}{Re}, \quad \theta = \frac{1-y}{2}$$

We trained our framework with 250 samples inside the domain and 30 samples on each boundary with 10000 iterations for this case. The training time for this case is 40 minutes. The training points are sampled using Latin hypercube sampling, and the loss function for this problem contains only the Dirichlet boundary condition loss for the velocity and the temperature on the walls combined with the residual loss inside the domain. After training, we performed a prediction on a (251×251) grid and obtained the velocity, pressure, and temperature fields. The predicted fields can be seen in Figure 4.1. The accuracy of the PINN is highly dependent on the weights of the loss function. In this case, we tried different weights of the different terms of the loss function to match our solution with the exact solution. Especially for an accurate pressure field, we increased the weights of the boundary condition losses. The solution in Figure 4.1 is obtained with a boundary loss weight ω_{BC}, which is eight times higher than the weight of the residual loss ω_R. Since the convective effects are not very dominant for this problem, boundary losses are dominant, so increasing the boundary loss weights increases the accuracy.

We test the performance of the PINNs with the addition of true observations at random points on the domain. We fused different numbers of randomly sampled exact solutions inside the domain to the network and add a data loss term into the loss function. The training process is done with 250 points inside the domain and 30 boundary points on each boundary beside the true solution points. In Table 4.1, we can see the L_2 norm of the error of the predicted u velocity and temperature fields. The number of observations represents the addition of the true solutions. Increasing this number

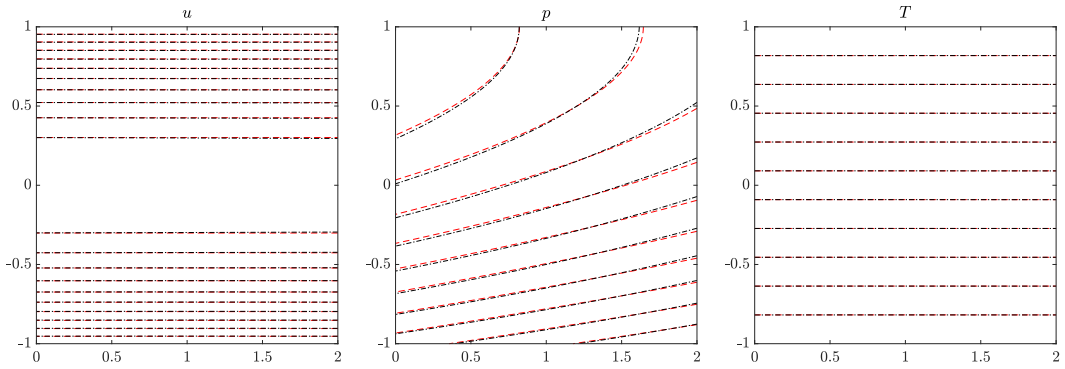


Figure 4.1: Prediction of the Poiseuille flow with PINN. The u velocity, pressure, and temperature fields are shown in order. Black contours show the exact solution, while the red dashed contours show the solution with PINN.

Table 4.1: L_2 norm of the error of the predicted u velocity and the temperature fields.

Number of Observations	u	T
20	0.527	0.087
50	0.253	0.057
100	0.166	0.034
150	0.141	0.032

reduces the L_2 norm of the prediction of the velocity and the temperature from the true solution.

Table 4.2: L_2 norm of the error of the predicted u velocity and the temperature fields with changing number of collocation points

Number of Collocation Points	u	T
250	0.3670	0.0252
500	0.3596	0.0571
1000	0.4054	0.0372

To show the effect of the number of sample points on the accuracy, a series of solutions are obtained with different numbers of fixed collocation points and also with

different batch sizes for dynamic sampling. In Table 4.2 the L_2 norms of the error of predicted u velocity and temperature fields are presented. Additionally, in Table 4.3, the same results are presented with changing batch sizes of dynamic sampling. In this study, we cannot observe any convergence rate with changing the number of points as is the case in traditional numerical methods. The convergence rate in PINN is related to its neural tangent kernel [45] and specific to the solved PDE.

Table 4.3: L_2 norm of the error of the predicted u velocity and the temperature fields with changing batch size with dynamic sampling strategy

Batch Size	u	T
128	0.3594	0.0598
256	0.3244	0.0388
512	0.4034	0.0522

4.2.2 Differentially Heated Cavity

We focus on the natural convection problem on a two-dimensional closed enclosure. The enclosure is a square cavity with its height denoted as $H = 1$ and width as $W = 1$. The boundary conditions of the cavity are simple no-slip walls, $u = 0,\ v = 0$, on all four walls. The thermal boundary conditions on the left and right walls are prescribed as

$$\theta_L = 1, \quad \theta_R = 0,$$

and the upper and the lower walls are thermally insulated

$$\frac{\partial \theta}{\partial y} = 0, \quad \text{for} \quad y = 0 \quad \text{and} \quad y = H.$$

The flow conditions are $Pr = 0.71$, and three different Rayleigh numbers as $Ra = 10^3,\ 10^4,\ 10^5$.

For the PINN solution, we sampled 150 points on each boundary and 1000 collocation points inside the domain for the training process. Boundary points are used to minimize the loss of Dirichlet and Neumann boundary conditions, and collocation

points are used to minimize the residual inside the domain. Automatic differentiation is used to calculate the derivatives on Neumann boundaries. All the experiments are done with 25000 iterations in roughly 1.5 hours.

Table 4.4: Maximum and minimum velocities along the center lines of the square cavity for $Pr = 0.71$ and $Ra = 10^3, 10^4, 10^5$.

	$Ra = 10^3$		$Ra = 10^4$		$Ra = 10^5$	
	u_{max}	v_{max}	u_{max}	v_{max}	u_{max}	v_{max}
PINN	0.137	0.138	0.192	0.233	0.128	0.258
Karakus [68]	0.137	0.139	0.192	0.233	0.129	0.257
Stokos et al. [75]	0.137	0.139	0.192	0.233	0.130	0.256
De Vahl Davis [76]	0.136	0.138	0.192	0.234	0.153	0.261

After the training process, prediction is performed on a (251×251) grid. In Table 4.4, we presented the maximum and minimum velocities on the horizontal and vertical centerlines after the prediction with PINN. For cases with different Ra numbers, our framework has values that are comparable with the ones in the literature. In Figure 4.2, the solution of PINN and its comparison with a high-fidelity solver through the temperature contours can be seen. Also, Figure 4.3 shows the center line profiles of velocity and temperature for different Ra numbers. These contours and profiles qualitatively match with the high-order solutions [68]. To increase the accuracy, we changed the weights of different loss terms as Ra changes. In Table 4.5, we presented the center line velocities for different Ra numbers and different weight ratios of the residual loss over the boundary loss where ω_R represents the weight of the residual loss, and ω_{BC} represents the weight of loss on the boundary conditions. We stopped changing weights when we matched the center line velocities with the reference solutions. As the Ra increases, the convective effects inside the domain become more dominant. Hence we need to decrease the weight of the boundary losses and focus more on the residual inside the domain. We select the loss ratio according to Table 4.5 that minimizes the error both inside the domain and on the boundaries. To obtain different solutions, the weight ratio is multiplied with two for different configurations and the maximum centerline velocities are compared.

Table 4.5: Maximum and minimum velocities along the center lines with different weight ratios of residual loss and the boundary loss.

ω_R/ω_{BC}	$Ra = 10^3$		$Ra = 10^4$		$Ra = 10^5$	
	u_{max}	v_{max}	u_{max}	v_{max}	u_{max}	v_{max}
0.5	0.137	0.138	0.190	0.231	0.137	0.273
1			0.192	0.233	0.128	0.258
2					0.132	0.261
4					0.130	0.261

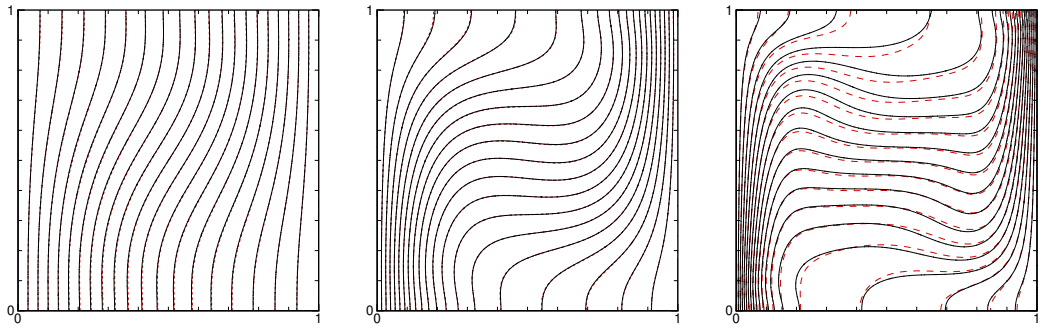


Figure 4.2: Temperature contours for the square cavity test. The high fidelity solution obtained with high fidelity discontinuous Galerkin solver between 1 and 0 with the increment of 0.05 for $Ra = 10^3, 10^4, 10^5$ from left to right shown with the black contours while the red dashed contours are the solution with PINNs.

We tested different types of neural network architectures and monitored the behavior of the total loss of the Adam optimizer for the cavity problem with $Ra = 10^3$ and presented in Figure 4.4. The plain fully connected network (FCN), a variation of the fully connected network named as Deep Galerkin Method (DGM) [77], a Fourier network and a modified Fourier network [78], a modified highway network using Fourier features [79], and a multiplicative filter network [80] are used. All of these architectures are readily available in the NVIDIA Modulus framework. In all the tests, 8-layer networks are constructed with 40 units. Hyperbolic tangent is set as the activation function, and the learning rate is 10^{-3}. The Fourier mapping architectures converge later than the plain fully connected network since the problem does not have

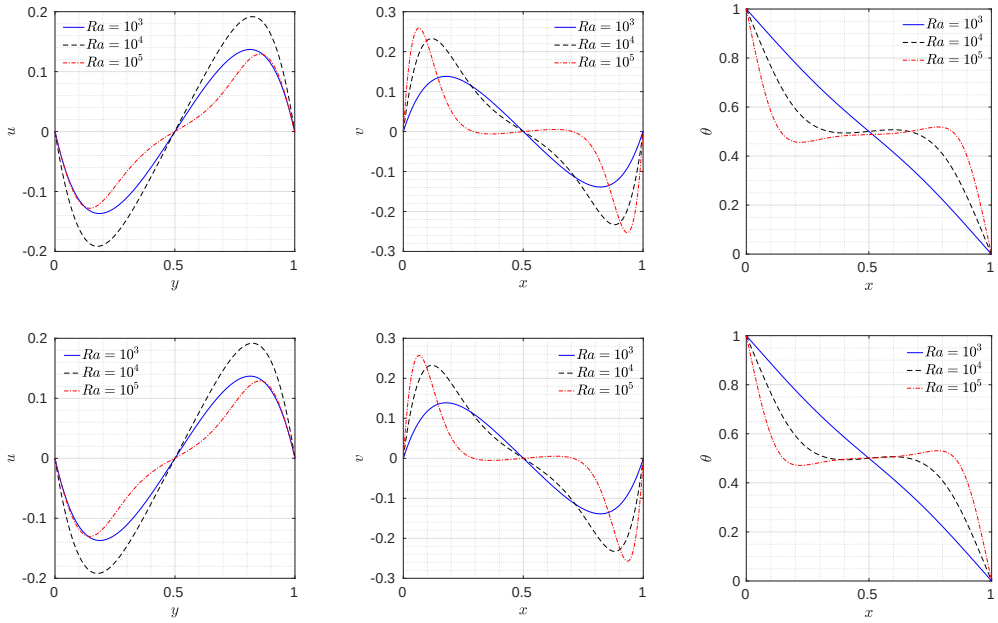


Figure 4.3: Velocity and temperature profiles along the $y = 0.5$ and $x = 0.5$ lines for different Ra numbers. The first row shows the values obtained with the PINN, while the second row shows the values of the high-order discontinuous Galerkin solver.

multi-scale behavior. For this simple problem, we do not need a Fourier mapping. Hence networks that are basically built on plain, fully connected networks converge in fewer iterations.

4.2.3 Heated Block

In this section, we focus on an application of coupled heat transfer with a partially blocked channel. The domain and the boundary conditions can be seen in Figure 4.5. The heated block represents an electronic part on a vertical electronic board [81]. The top wall is adiabatic, and the bottom wall is at a prescribed temperature. A low-temperature flow comes from the inlet, and the outflow is a fully developed outlet meaning the changes in the x direction is zero. The Prandtl number is set to 0.7 for this problem and the Reynolds number is 37.8. The ratio of Gr/Re^2 is 1 and the forcing is in the x direction.

For training the network, we sampled 30 points on the inlet and the outlet, 210 points

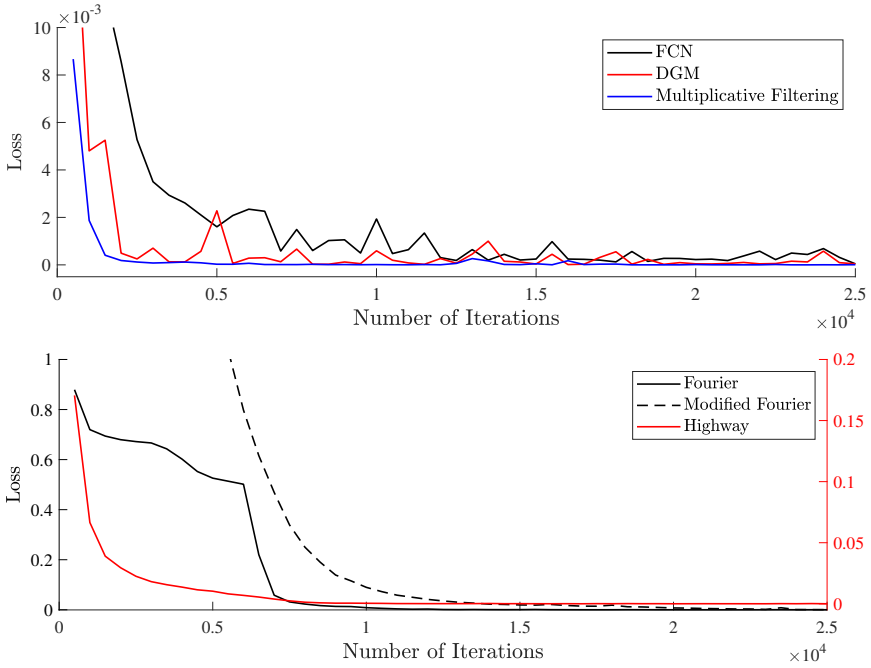


Figure 4.4: Behavior of the total loss in the square cavity problem with $Ra = 10^3$ with different types of neural network architectures.

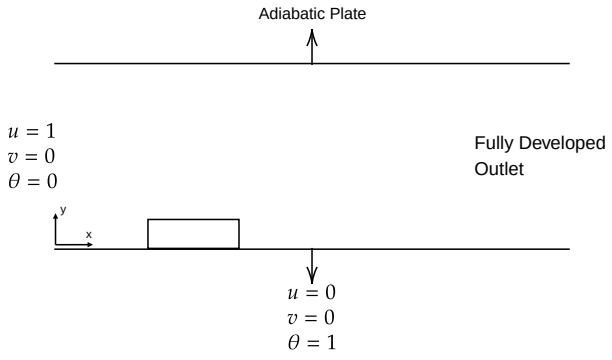


Figure 4.5: Schematic of the partially blocked channel

on the adiabatic wall, 40 points on the heated block, 180 points on the bottom wall, and 1400 points inside the domain. We used 25000 iterations for the Adam optimizer with the learning rate of 5×10^{-4} and obtained the solution presented in Figure 4.6. PINN solution well predicts the Neumann boundary conditions on the top wall, the fully developed outlet, and the no-slip Dirichlet velocity conditions.

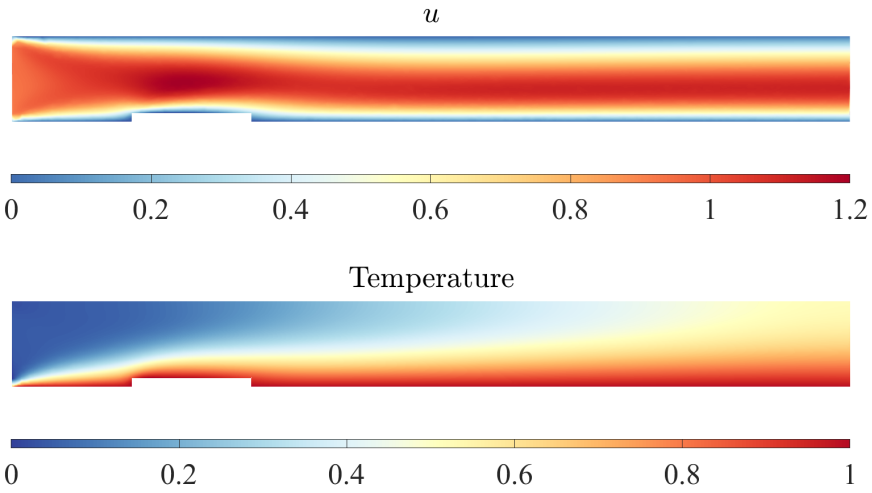


Figure 4.6: Velocity and temperature profiles for the heated block case. The figure on the top shows the velocity profile and the figure on the bottom shows the temperature field predicted by the PINN.

CHAPTER 5

PHYSICS-INFORMED NEURAL NETWORKS FOR MESH DEFORMATION WITH EXACT BOUNDARY ENFORCEMENT

Dynamic grids in numerical fluid flow simulations generally arise in many applications, such as airfoil movement [82, 83], blood flow [84], parachute mechanics [85, 86], and free surface flow problems [87]. These and other fluid-structure interaction (FSI) problems need to move the computational grid with moving boundaries. The naive choice is to regenerate the mesh every time the boundary moves. Regenerating the mesh for a complex geometry results in a need for an automatic mesh generator [88]. This approach alters the grid connectivity and, therefore, brings up a need to project the solution to the new mesh. This introduces new projection errors each time the mesh is updated. Moreover, the cost of calling a new mesh generation algorithm can be overwhelming, especially for 3D problems [89].

Specific mesh moving techniques can overcome the drawbacks of remeshing for moving boundary problems. These methods try to update the position of the nodes of the original mesh under some prescribed laws without changing the grid connectivity. Farhat et al. introduced a spring analogy, where they fictitiously insert a torsional spring to the nodes of the mesh [90]. The system has fictitious mass, damping, and stiffness matrices, and the forcing is the displacement of the moving boundaries. This approach prevents vertex collisions as well as penetrating grid edges. In [89], the authors used a linear elastic equation to represent the fluid domain as an elastically deformable body and introduced a parallel finite element strategy. Using the same elasticity formulation, Stein et al. [3] solved the equation using a Jacobian-based stiffening. They introduced an additional stiffening power as a function of transformation Jacobian in the finite element formulation. This addition allowed them to

stiffen the smaller elements more than the larger ones, resulting in improved mesh quality near the moving surfaces. Takizawa et al. [91], introduced a method based on the fiber-reinforced hyperelasticity model. They introduced fibers in different directions according to the motion, which allows the model to reduce the distortion of a mesh element. The moving mesh problem can be solved using the Laplacian or biharmonic equations [92, 93, 94]. Although using the biharmonic operator introduces extra computational complexity compared to the Laplacian equation systems, it can give the extra ability to control the normal mesh spacing [94].

Despite the success of PINN across a range of different problems, it can face difficulties when solving multiscale and multiphysics problems [12], especially for dynamical systems with chaotic or turbulent behavior [95]. The fully connected networks face difficulties in learning high-frequency functions. This phenomenon is named spectral bias [47, 45]. The high-frequency behavior in the objective function results in sharp gradients. Therefore, PINN models can have difficulties while penalizing the residual loss. Although there are several approaches to tackle these problems and improve the training capabilities of PINN, the classical PINN method shows better performance to accurately solve the PDEs that govern the mesh deformation. Therefore our research focuses on using PINNs in the application of these problems.

The main objective of this chapter is to show the applicability of physics-informed neural networks for moving mesh problems. The PINN approach can produce satisfactory solutions for the movement of boundaries without needing a discretization scheme. However, using the original PINN formulation for mesh moving problems can have difficulties with the static and moving boundaries. PINN minimizes the loss at the boundaries, without imposing boundary conditions exactly. To overcome this problem, we used exact boundary enforcement. After obtaining a particular solution that weakly satisfies the boundary conditions, the prediction is corrected by training another PINN. To the best of our knowledge, using PINNs on mesh movement problems with exact boundary enforcement is not studied in detail in the literature.

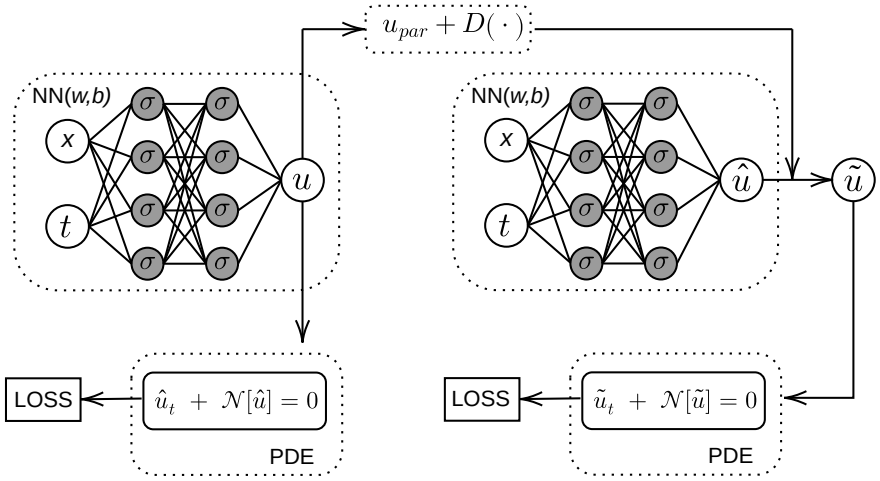


Figure 5.1: Schematic of PINN approach with exact boundary enforcement. The first PINN on the left shows the original formulation with weakly enforced Dirichlet boundary conditions. The second network uses the particular solution with exact boundary enforcement to satisfy Dirichlet boundaries exactly

5.1 Exact Boundary Enforcement

The optimization algorithm used in PINN tries to minimize the physics-based loss, $\mathcal{L}_R$. Using proper boundary and initial conditions can regularize the physics loss in deep neural networks. This classical PINN boundary condition implementation is named soft boundary enforcement [96]. In this approach, the boundary prediction is minimized in the composite loss function. Although the SGD algorithms can minimize these loss functions they do not satisfy the boundary conditions exactly. However, some PDE applications, such as mesh movement, need exact boundary values. For this purpose, we apply exact boundary enforcement. First, we trained a PINN with soft boundaries. For the mesh movement problem, the displacement vector $\mathbf{u} = [X, Y]^T$ will give the new coordinates of the nodes from the first neural network prediction. This solution is then changed on the boundaries with the exact values. This new solution is the particular solution of our approach. Then, a new PINN is trained with an output $\hat{\mathbf{u}}(\mathbf{x}; \theta)$. This output is modified with the following equation.

$$\tilde{\mathbf{u}}(\mathbf{x}; \theta) = \mathbf{u}_{par}(\mathbf{x}) + D(\mathbf{x})\hat{\mathbf{u}}(\mathbf{x}; \theta). \tag{5.1}$$

Here, $\mathbf{u}_{par}$ is a particular solution that is a globally defined smooth function that only satisfies the boundary conditions. Any smooth function can be used for the particular solution such as radial basis functions (RBF) or linear functions [96]. In this work, we use the classical PINN predictions with the soft boundary condition implementation as the particular solution. D is a specified distance function from the boundary. Equation 5.1 states that on the boundaries $D(\mathbf{x}) = 0$, the particular solution satisfies the exact boundary values, $\mathbf{u} = g$ on $\partial\Omega$. For a general approach, we used the shortest distance between the residual points and the boundaries. Since the geometries in this study are not too complex, this approach is not consuming much time. For complex geometries, approximate distance functions using R-functions [97] or pre-trained deep neural networks [98] can be used. This modified output contributes to the physics loss of the new PINN. In this network, the objective function is only consisting the PDE residual $\mathcal{L}_R$ and trained with the same PDE. This approach allows us to exactly satisfy the Dirichlet boundary conditions using PINN and the schematic can be seen in 5.1.

5.2 Mesh Deformation

Mesh movement strategies to deform the mesh with a moving boundary generally can be performed by solving a PDE or using an interpolation scheme [99]. All of these techniques have the goal to provide a displacement of the moving boundary and propagate this movement into the domain. Methods with a PDE solution, generally model the domain as a physical process which can be solved using numerical methods. One of the popular versions includes modeling the domain with torsional springs that prevent the vertices to collide [90]. In a similar manner, this movement can be modeled with an elastic [89, 3] and hyperelastic [91] analogy, where the computational domain is simulated as an elastic body. Nonlinear elasticity equations with neo-Hookean models can be used in the same way as the elastic equations [100]. Other techniques include mesh deformation as a diffusive system modeled with the Laplacian or biharmonic equations [94]. All these PDEs can be solved using traditional numerical methods such as FEM.

Interpolation schemes consider the mesh movement as a problem of interpolation

from the boundaries to the domain. These schemes use interpolation on scattered data and generally do not need connectivity information. Using radial basis functions (RBF) is one of the common methods. In [101], de Boer et al. use RBF interpolation on unstructured grids to estimate the movement. The equation system only involves the boundary nodes and displacement of the whole mesh is modeled. Extending this method, in [102], the authors use data reduction algorithms using a coarse subset of the surface mesh. With greedy algorithms, this approach is effective, especially for mesh motion problems with smooth surface deformations.

In this work, we used one of the common PDEs for mesh movement. The mesh motion is calculated by using the linear elasticity equation from structural mechanics. The coordinates of the nodes will be defined as $\mathbf{u}$, the computational domain is referred to as Ω, and the boundaries are $\partial\Omega$. Boundaries also include the moving objects inside the meshes. The new coordinates of the moving and stationary boundaries are given as the Dirichlet boundary condition. The movement of an object inside the mesh deforms the computational domain which is modeled as an elastic body. The new coordinates can be found by the following linear elasticity equation:

$$\nabla \cdot \boldsymbol{\sigma}(\mathbf{u}) = 0 \quad \text{in } \Omega \tag{5.2a}$$

$$\mathbf{u} = \mathbf{u}_b \quad \text{on } \partial\Omega. \tag{5.2b}$$

Here $\boldsymbol{\sigma}$ is the Cauchy stress tensor. It is related to the strain tensor $\boldsymbol{\epsilon} = (\nabla\mathbf{u} + \nabla\mathbf{u}^T)/2$. The stress tensor can be written in a way by Hooke's law:

$$\boldsymbol{\sigma} = \lambda \mathrm{tr}(\boldsymbol{\epsilon})\mathbf{I} + 2\mu\boldsymbol{\epsilon}. \tag{5.3}$$

The Lamé parameters λ and μ are structural parameters coming from the elastic modulus E and Poisson's ratio ν. Since the mesh domain is not a real elastic body, the exact values for these parameters are not known. A value between 0.3 and 0.45 is recommended for Poisson's ratio [100] since a high value can lead to distorted elements, and a lower value can reduce the resistance.

To be able to compare the effectiveness of different mesh movement techniques after a deformation, we use a mesh quality metric based on [3]. In these metrics, the area and shape changes are considered by checking the element area and the aspect ratio.

Both metric uses the initial mesh elements as reference elements and measures the change according to them. The element area change f_A^e and shape change f_{AR}^e is defined as :

$$f_A^e = \left| \log\left(\frac{A^e}{A_o^e}\right) / \log(2.0) \right|, \tag{5.4a}$$

$$f_{AR}^e = \left| \log\left(\frac{AR^e}{AR_o^e}\right) / \log(2.0) \right|. \tag{5.4b}$$

Here, the superscript e represents the specific element, and the subscript o is the initial mesh element before the deformation occurs. AR^e is the element aspect ratio defined in [3] as:

$$AR^e = \frac{(l_{max}^e)^2}{A^e}. \tag{5.5}$$

Here, l_{max}^e is the maximum edge length for the specific element. For comparison of different techniques, we use the global area and shape changes by considering the maximum values of element area and shape changes, respectively.

5.3 Results

The movement of dynamic meshes with PINN is presented with several different test cases. First, a deformed square is presented where we squeeze the domain from the top and bottom. Then, the basic translation and rotation tests are performed and the solutions of the PINN approach are compared with the finite element solutions. Lastly, the movement of a flexible beam is presented where one end of the beam is fixed. For all the problems, initial meshes are generated using Gmsh mesh generator [103]. We used TensorFlow to construct our PINN framework with Adam optimizer as the gradient descent algorithm. We initialized all the neural networks using the Glorot scheme and used 7 hidden layers with 50 units. The classical neural networks are trained for 40000 iterations, and the networks with exact boundary enforcement are trained for 5000 iterations. The learning rate is 10^{-3} with a decay rate of 0.9. Training time for all the experiments are roughly 25 minutes for classical neural networks and 3 minutes for the exact boundary enforcement.

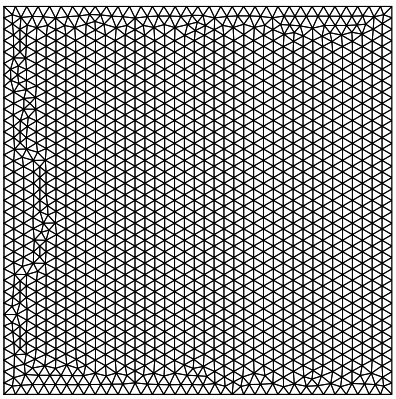
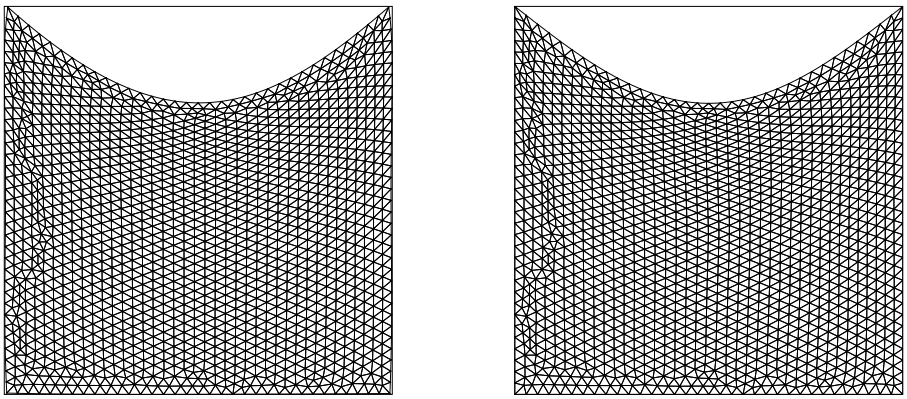


Figure 5.2: The initial unstructured mesh consists of 2744 triangular elements.

5.3.1 Deformed Square

In this test case, a square domain is deformed from its boundaries. The square domain is $x, y \in [0, 1] \times [0, 1]$ and the unstructured mesh consists of 2744 triangular elements. The initial mesh can be seen in Figure 5.2. We want to find a deformed mesh where the position of the top boundary becomes $\hat{y} = y - 0.25 \sin(\pi x)$. On the top surface, we implement this condition as a Dirichlet boundary condition as well as $\hat{x} = x$. All the other boundaries have the same Dirichlet boundary condition as $\hat{y} = y$, and $\hat{x} = x$. The deformed mesh can be seen in Figure 5.3. The figure in the middle shows the results obtained by only using classical PINN. This shows the boundaries are not in the exact position and are deformed in an undesired way. The figure on the right shows the solution after exact boundary enforcement. The boundary values are corrected with the exact positions with the proposed approach. The L_2 error on the boundary nodes is calculated as 0.031. For this test case, we increased the specific weight of the boundary loss of the composite loss function. Since the deformation of the boundary is higher than the deformation of the computational domain, the boundary weight is increased. The weight ratio of the boundary loss and the residual loss is set to 25 to capture the boundary values more precisely.

The mesh quality measure of the deformed mesh based on the element area and shape changes can be seen in 5.4. The top surface is deformed according to a sinusoidal

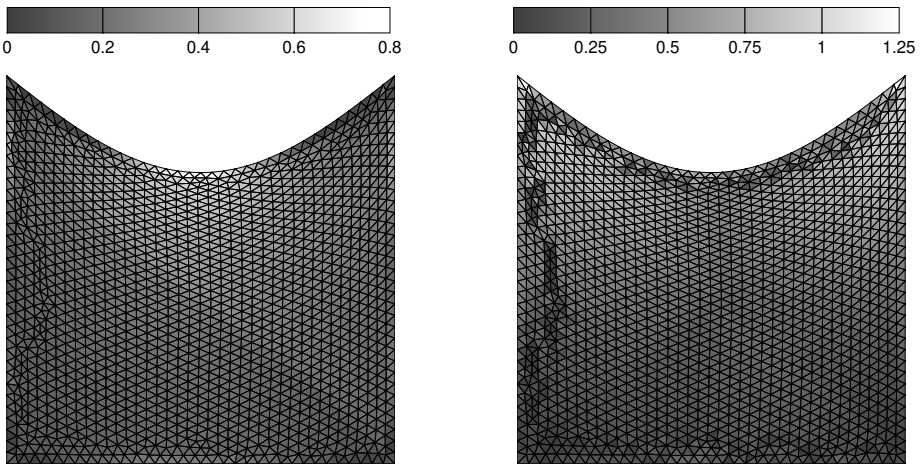


(a) (b)

Figure 5.3: Deformed square case with its deformed top boundary. The first deformed figure (a) shows the solution with classical PINN. Figure (b) represents the solution with exact boundary enforcement.

function. The elements near the deformed boundary have the most change in size and shape as expected. Especially in the middle where the deformation is the largest, the elements are squeezed and get smaller. In the corners where the element vertices have two boundary conditions in each direction, the element area change is not significantly large. However, the shape of the corner elements changes more than the other elements on the boundary. These elements are bounded by the two boundaries and therefore the aspect ratios get larger. The deformation of the inner elements is relatively low, especially near the bottom boundary. The mesh deformation values get lower as the elements' position is away from the deformed boundary. The global area and shape change metrics are calculated as $|f_A^\infty| = 0.744$, $|f_{AR}^\infty| = 1.264$, respectively.

To see the capabilities of our approach we further deform the bottom boundary with its coordinates $\hat{y} = y + 0.25\sin(\pi x)$. The Dirichlet boundary conditions on the other boundaries are the same as the other, $\hat{x} = x$, $\hat{y} = y$. The same specific weight ratio for the loss function of the PINN formulation is used. The deformed configuration can be seen in Figure 5.5. The figure in the middle is the solution with the classical



(a) Area change

(b) Shape change

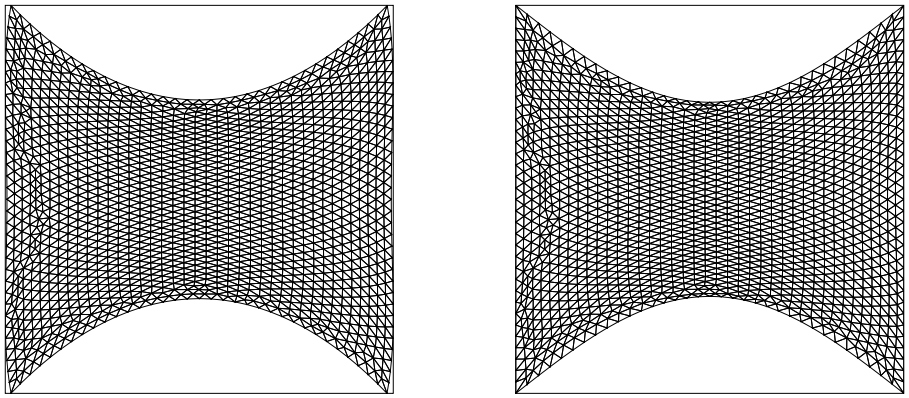
Figure 5.4: Element quality metrics of the square with deformed top boundary. The figure on the left shows the element area change and the figure on the right shows the element shape change with respect to the initial mesh elements.

PINN approach. The vertices on the boundaries are not in exact positions. Especially on the corners, the classical PINN solution has difficulty satisfying the positions. The L_2 error of the boundary positions is calculated as 0.076 for this case.

The elementwise quality measures of this case can be seen in Figure 5.6. Same as the case with only top boundary deformation, the elements on the top and the bottom boundaries are deformed the most. The elements in the middle collapsed more than the case before. The global area and shape change values are $|f_A^\infty| = 1.701$, $|f_{AR}^\infty| = 1.845$, respectively. The element shape and size change significantly as the deformation is increased.

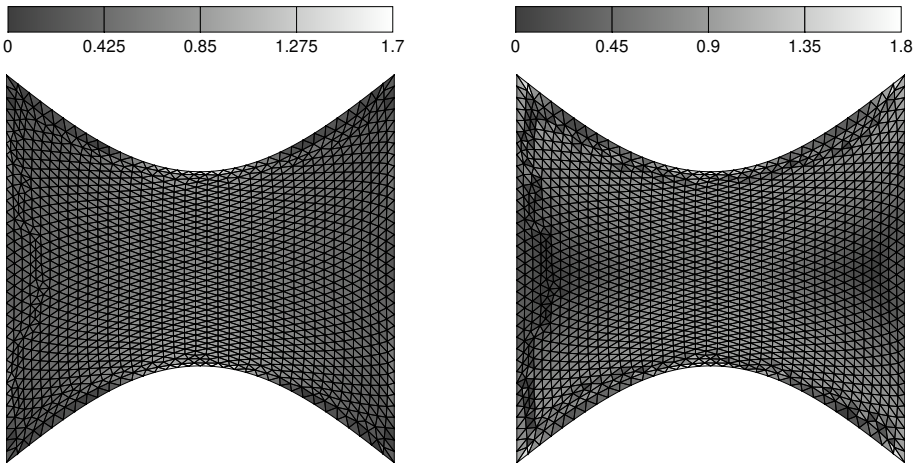
5.3.2 Translation and Rotation tests

To test the accuracy of our approach, classical translation and rotation tests in [3] is performed. The original mesh can be seen in Figures 5.8 and 5.9. There is a line object located in $(-L, 0) \times (L, 0)$ in a $(-2L, -2L) \times (2L, 2L)$ domain. A total of 2182 triangles are generated for the mesh.



(a) (b)

Figure 5.5: The deformed square is squeezed from its top and bottom boundary. Figure (a) shows the solution with classical PINN. Figure (b) represents the solution with exact boundary enforcement.



(a) Area change (b) Shape change

Figure 5.6: Element quality metrics of the square deformed from the top and bottom boundaries. The figure on the left shows the element area change and the figure on the right shows the element shape change with respect to the initial mesh elements.

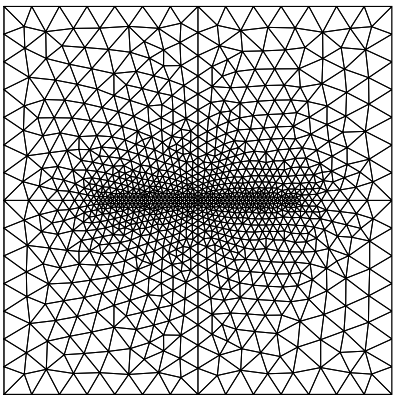
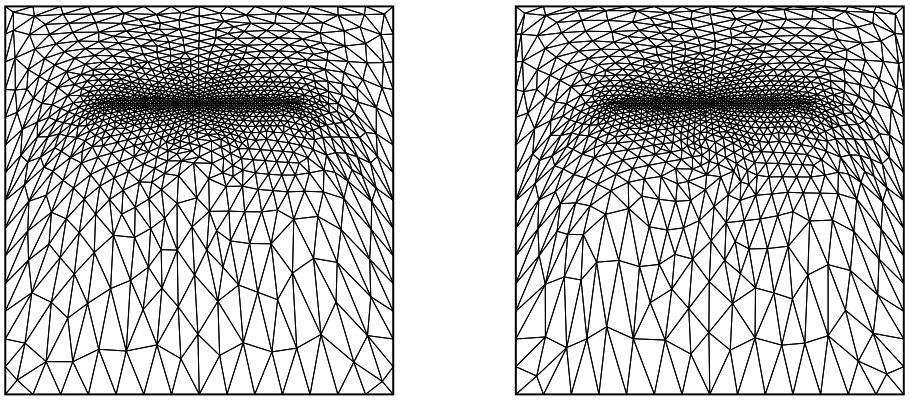


Figure 5.7: Initial unstructured mesh with a total of 2182 triangular elements.

For the translation tests, the object is moved $0.5L$ upwards. The movement is performed in 10 steps with $0.05L$ and in 5 steps with $0.1L$ movement upwards in two different training settings. The last step of the movement can be seen in Figure 5.7. In Figure 5.10, the PINN method is compared with the approach in [3]. The area and shape change metrics of two PINN solutions are presented alongside the classical finite element solutions and solutions with Jacobian-based stiffening. The authors applied a stiffening power to prevent the deformation of the smaller elements. The stiffened approach represents the best value obtained in [3] with different applied stiffening power. The two PINN solutions are representing the overall motion in 5 and 10 steps. The total number of steps is represented in parentheses in the figure. As seen in the first row of Figure 5.10, the PINN solutions are comparable with the FEM solutions with Jacobian-stiffening. As mentioned before, the PINN approach does not have any criteria to prevent mesh overlapping and sudden movements move the vertex nodes in an undesired way. Therefore, the quality of the deformed mesh improves as the number of steps increases.

For the rotation tests, the object is rotated 0.25π counterclockwise. Again to prevent overlapping of edges and collision of vertices, the movement is performed in steps with 0.025π and 0.05π counterclockwise movement in two different training. The last step of the rotation can be seen in Figure 5.9. The deformed mesh differs especially on the boundaries between different PINN solutions. The small elements near the



(a) (b)

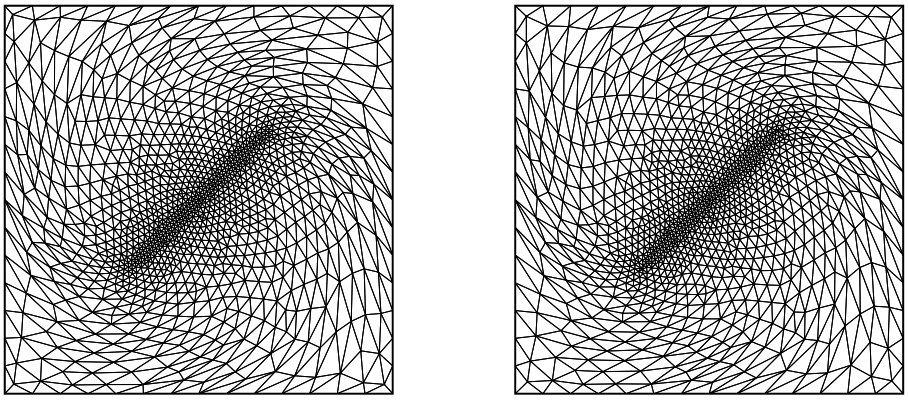
Figure 5.8: Deformed mesh after a total translation of 5 units. The solution in figure (a) is performed in 10 steps while the solution in figure (b) is performed in 5 steps.

Table 5.1: Global area and shape changes of translation tests. The solution is performed in 10 steps. The values are given in every step.

Δy	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
$\lvert f_A \rvert_\infty$	0.67	1.20	1.19	1.29	1.52	1.63	1.70	1.72	1.79	2.00
$\lvert f_{AR} \rvert_\infty$	0.60	1.13	1.12	1.22	1.45	1.56	1.63	1.66	1.92	2.26

moving boundary start to collapse in the PINN solution with 5 steps. As the step number increases, the mesh quality increases. The comparison of the rotation tests with the same finite element solution of the translation tests is presented in Figure 5.10. The PINN approach again lies between the classical solution and the solution with Jacobian-based stiffening.

In both tests, the global mesh quality metric presented in section 5.2 is used. The $|f_A|_\infty$ and $|f_{AR}|_\infty$ are calculated as the maximum area and shape change of the values in Equation 5.4 in every step. The area change and shape change values are presented in Tables 5.1 and 5.2, for the translation and rotation tests, respectively.



(a) (b)

Figure 5.9: Deformed mesh after a total rotation of 0.25π. The solution in figure (a) is performed in 10 steps while the solution in figure (b) is performed in 5 steps.

Table 5.2: Global area and shape changes of rotation tests. The solution is performed in 10 steps. The values are given in every step.

$\Delta\theta(\pi)$	0.025	0.05	0.075	0.1	0.125	0.15	0.175	0.2	0.225	0.25
$\lvert f_A\rvert_\infty$	0.27	0.43	0.66	0.88	0.88	1.15	1.09	1.26	1.30	1.32
$\lvert f_{AR}\rvert_\infty$	0.36	0.51	0.75	1.03	1.27	1.55	1.91	2.14	2.52	2.57

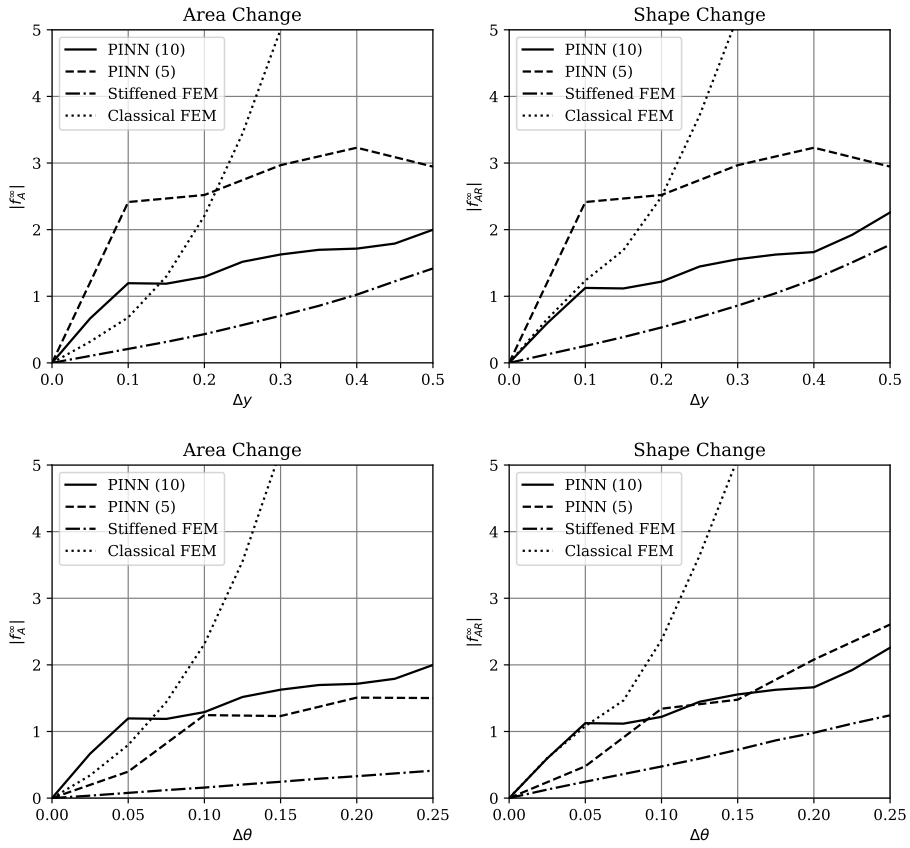


Figure 5.10: Global area and shape change metrics of the translation and rotation tests compared with the FEM solution in [3]. The first row shows the comparison of the translation test, while the second row shows the comparison of the rotation tests.

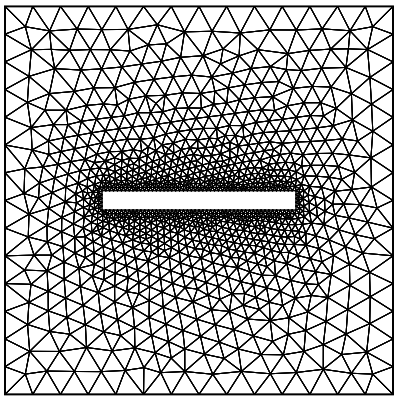
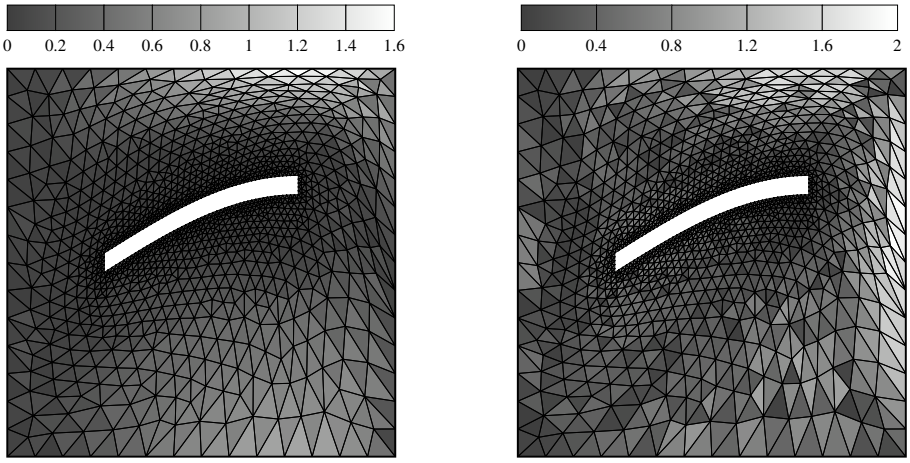


Figure 5.11: Initial mesh of the flexible beam test case with 2098 triangular elements. The elements are concentrated on the moving boundary to track the deformation in a precise way.

5.3.3 Flexible Beam

This test case consists of a mesh movement due to a motion of a flexible beam adapted from the problem in [100]. The beam is fixed on its left end and sits in the center of the domain. Domain dimensions are $(-10, 10) \times (-10, 10)$ and the structure's position is $(-5, 5) \times (-0.5, 0.5)$ The deformation is based on a sinusoidal function $\sin(\frac{\pi}{2}\frac{x}{L})$ with varying amplitude. The initial mesh can be seen in Figure 5.11. This unstructured mesh consists of 2098 triangular elements. The right end of the structure first moves to 4 units upwards, then 8 units downwards, following a 4-unit upward motion to return to its initial state. The movements are performed step by step with a 2-unit motion, upwards or downwards.

In Figure 5.12, the deformed mesh after two steps of movement is presented with the mesh quality presented with the global area and shape change metric. Using exact boundary enforcement gives the true boundary position and therefore fixes the vertices on the boundaries. Therefore, on the outer boundaries, elements are stretched and squeezed more than the inner elements. Especially elements near the tip of the moving boundary have the most area and shape changes.

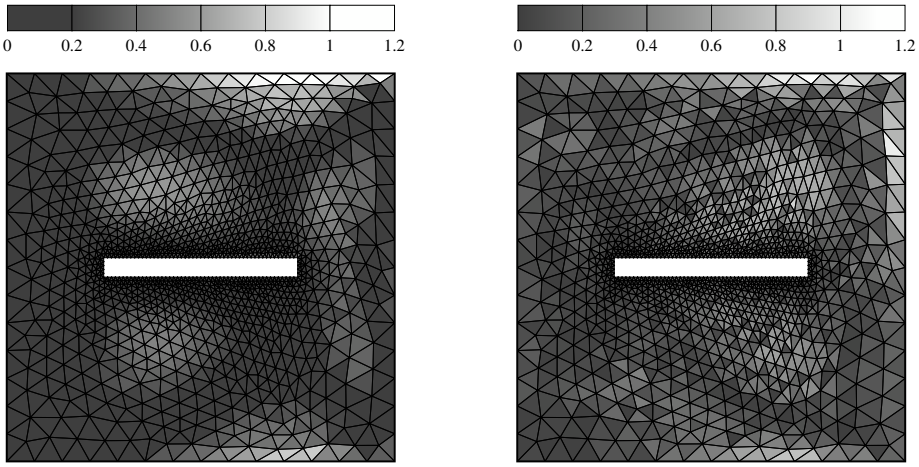


(a) Area change

(b) Shape change

Figure 5.12: Element quality metrics when the structure tip moves to $y = 4$.

In figure 5.13, the mesh after one cycle of motion is presented. The structure returns to its original place after eight steps. By looking at the area change the sinusoidal motion of the structure can be observed. The most deformed elements are located at the top and bottom boundaries and near the moving tip of the structure. These elements are squeezed first and cannot recover themselves after the relative stretching.



(a) Area change

(b) Shape change

Figure 5.13: Element quality metrics when the structure returns its original position.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

In this research, physics-informed neural networks are analyzed in the context of computational fluid dynamics applications. These networks learn from the nature of the considered partial differential equation. The networks are used to get direct solutions to CFD problems as well as the solutions for complementary equations needed in CFD. The accuracy and convergence rate of the networks are further increased by changing the network parameters and adding labeled data from high-fidelity solutions.

PINN methodology minimizes a physics-informed loss function defined on a computational domain. This minimization problem often gets stuck on a local minimum and cannot converge to a physically meaningful solution. The current formulation of PINN allows getting solutions of partial differential equations on simple domains and equations that do not have multiscale behavior. Therefore, the presented differential equations in this research cover the solutions to flow problems in low Reynolds number regimes. As the Reynolds number increases, the nonlinearity of the solution increases, and PINN formulation faces difficulties. Because of that for the Navier-Stokes equations, the solution of a lid-driven cavity at $Re = 100$ is presented. Classical PINN formulation can capture the trend of the flow but struggles to get an accurate solution on the corner discontinuity. To show the convergence, the plot of the PINN loss is presented in the lid-driven cavity solution. The residual loss starts from a very low number indicating the trivial solution. The information on the boundary then propagates and the loss value jumps to a higher value. It is shown that the trend of the boundary and the residual loss plots are similar because the propagation of the information is from the boundary to the domain. In the context of Euler equations, a con-

tact discontinuity is captured with PINN. The problem is a simple one-dimensional problem and PINN works well to get an accurate solution. However, around the discontinuity, the solution oscillates. For capturing sharp gradients we imposed additional labeled data from the analytical solution. This addition improves the accuracy around the discontinuous region as well as the overall accuracy.

The weights in each loss term in the overall composite loss function can change the solution since each term affects differently to the convergence of PINN. In the context of thermal convection problems, it is presented that the weight of the boundaries is important when the flow is in a low Rayleigh number regime. This effect decreases as the Rayleigh number increases. As the flow regime gets complicated the learning capacity of PINN decreases. Therefore, in the literature, the research focuses on improving the learning dynamics of PINN.

Dirichlet boundary conditions are learned as supervised learning since these labeled data are supplied to the network. The optimization algorithm minimizes this supervised loss up to a threshold. However, in some problems, such as mesh deformation, the boundary values should be exact. For these types of problems, exact boundary enforcement is used. A particular solution is constructed with a classical PINN formulation. Then, the boundary values are corrected to the exact values and trained again with only PDE residual. This approach ensures the boundaries are in their true positions.

As mentioned many times earlier in this research, the current PINN formulation is suitable for problems that do not have multiphysics or multiscale behavior. Therefore, the partial differential equations used in this thesis show elliptic behavior or very little nonlinearity. PINN research is still a work in progress to solve challenging real-life problems. Instead of a replacement, using PINN as a complementary tool to high-order solvers can be a good practice. This idea is the direction of future works such that the problems without a need for high-order accuracy may be investigated. The mesh deformation study in this thesis can be extended with additional constraints to keep the mesh quality high. The mesh quality parameter can be a variable in the PINN formulation. Moreover, the identification of the amount and location of the artificial diffusion in nonlinear hyperbolic equations can be represented by PINN formulation.

Absorbing boundaries for wave propagation problems can also be future work.

REFERENCES

[1] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods*, vol. 54 of *Texts in Applied Mathematics*. New York, NY: Springer New York, 2008.

[2] U. Ghia, K. N. Ghia, and C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, pp. 387–411, Dec. 1982.

[3] K. Stein, T. Tezduyar, and R. Benney, "Mesh moving techniques for fluid-structure interactions with large displacements," *J. Appl. Mech.*, vol. 70, no. 1, pp. 58–63, 2003.

[4] J. H. Ferziger, M. Perić, and R. L. Street, *Computational Methods for Fluid Dynamics*, vol. 3. Springer, 2002.

[5] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, "Meshless methods: An overview and recent developments," *Computer Methods in Applied Mechanics and Engineering*, vol. 139, pp. 3–47, Dec. 1996.

[6] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: Theory and application to non-spherical stars," *Monthly notices of the royal astronomical society*, vol. 181, no. 3, pp. 375–389, 1977.

[7] S. Adami, X. Y. Hu, and N. A. Adams, "A generalized wall boundary condition for smoothed particle hydrodynamics," *Journal of Computational Physics*, vol. 231, no. 21, pp. 7057–7075, 2012.

[8] B. Nayroles, G. Touzot, and P. Villon, "Generalizing the finite element method: Diffuse approximation and diffuse elements," *Computational Mechanics*, vol. 10, no. 5, pp. 307–318, 1992.

[9] P. Breitkopf, A. Rassineux, J. M. Savignat, and P. Villon, "Integration con-

straint in diffuse element method," *Computer Methods in Applied Mechanics and Engineering*, vol. 193, pp. 1203–1220, Mar. 2004.

[10] W. K. Liu, S. Jun, and Y. F. Zhang, "Reproducing kernel particle methods," *International Journal for Numerical Methods in Fluids*, vol. 20, no. 8-9, pp. 1081–1106, 1995.

[11] R. Salehi and M. Dehghan, "A moving least square reproducing polynomial meshless method," *Applied Numerical Mathematics*, vol. 69, pp. 34–58, 2013.

[12] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, 2021.

[13] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[14] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[15] M. Kennedy and A. O'Hagan, "Predicting the output from a complex computer code when fast approximations are available," *Biometrika*, vol. 87, no. 1, pp. 1–13, 2000.

[16] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA, 2006.

[17] D. Duvenaud, *Automatic Model Construction with Gaussian Processes*. Doctor of Philosophy, University of Cambridge, 2014.

[18] P. Perdikaris, D. Venturi, J. O. Royset, and G. E. Karniadakis, "Multi-fidelity modelling via recursive co-kriging and Gaussian–Markov random fields," *Proc. R. Soc. A.*, vol. 471, no. 2179, 2015.

[19] L. Parussini, D. Venturi, P. Perdikaris, and G. Karniadakis, "Multi-fidelity Gaussian process regression for prediction of random fields," *Journal of Computational Physics*, vol. 336, pp. 36–50, May 2017.

[20] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Inferring solutions of differential equations using noisy multi-fidelity data," *Journal of Computational Physics*, vol. 335, pp. 736–746, Apr. 2017.

[21] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis, "Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling," *Proc. R. Soc. A.*, vol. 473, no. 2198, 2017.

[22] S. Lee, F. Dietrich, G. E. Karniadakis, and I. G. Kevrekidis, "Linking Gaussian process regression with data-driven manifold embeddings for nonlinear data fusion," *Interface Focus.*, vol. 9, no. 3, p. 20180083, 2019.

[23] B. Hamzi, R. Maulik, and H. Owhadi, "Simple, low-cost and accurate data-driven geophysical forecasting with learned kernels," *Proc. R. Soc. A.*, vol. 477, no. 2252, p. 20210326, 2021.

[24] H. Owhadi and G. R. Yoo, "Kernel Flows: From learning kernels from data into the abyss," *Journal of Computational Physics*, vol. 389, pp. 22–47, 2019.

[25] M. Raissi and G. Karniadakis, "Deep multi-fidelity Gaussian processes," *arXiv: 1604.07484*, 2016.

[26] X. Meng and G. E. Karniadakis, "A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems," *Journal of Computational Physics*, vol. 401, p. 109020, 2020.

[27] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[28] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[29] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5595–5637, 2017.

[30] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *Journal of Heat Transfer*, vol. 143, no. 6, 2021.

[31] X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations," *Journal of Computational Physics*, vol. 426, p. 109951, 2021.

[32] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[33] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 360, p. 112789, 2020.

[34] Q. Lou, X. Meng, and G. E. Karniadakis, "Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation," *Journal of Computational Physics*, vol. 447, p. 110676, 2021.

[35] S. Wang and P. Perdikaris, "Deep learning of free boundary and stefan problems," *Journal of Computational Physics*, vol. 428, p. 109914, 2021.

[36] A. L. Blum and R. L. Rivest, "Training a 3-node neural network is NP-complete," *Neural Networks*, vol. 5, no. 1, pp. 117–127, 1992.

[37] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, 2020.

[38] A. D. Jagtap and G. E. Karniadakis, "Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations," *Communications in Computational Physics*, vol. 28, no. 5, pp. 2002–2041, 2020.

[39] E. Kharazmi, Z. Zhang, and G. E. M. Karniadakis, "hp-VPINNs: Variational physics-informed neural networks with domain decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113547, 2021.

[40] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *Journal of Computational Physics*, vol. 404, p. 109136, Mar. 2020.

[41] A. D. Jagtap, K. Kawaguchi, and G. Em Karniadakis, "Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, p. 20200334, July 2020. Publisher: Royal Society.

[42] C. L. Wight and J. Zhao, "Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks," *arXiv preprint arXiv:2007.04542*, 2020.

[43] A. Daw, J. Bu, S. Wang, P. Perdikaris, and A. Karpatne, "Mitigating propagation failures in PINNs using evolutionary sampling," Oct. 2022. arXiv:2207.02338 [cs].

[44] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[45] S. Wang, X. Yu, and P. Perdikaris, "When and why PINNs fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022.

[46] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[47] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *International Conference on Machine Learning*, pp. 5301–5310, PMLR, 2019.

[48] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, pp. 121–136, Sept. 1975.

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

[50] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," Oct. 2017. arXiv:1710.05941 [cs].

[51] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010. ISSN: 1938-7228.

[52] S. Ruder, "An overview of gradient descent optimization algorithms," June 2017. arXiv:1609.04747 [cs].

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986. Number: 6088 Publisher: Nature Publishing Group.

[54] J. Lorraine and D. Duvenaud, "Stochastic hyperparameter optimization through hypernetworks," Mar. 2018. arXiv:1802.09419 [cs].

[55] K. You, M. Long, J. Wang, and M. I. Jordan, "How Does Learning Rate Decay Help Modern Neural Networks?," Sept. 2019. arXiv:1908.01878 [cs, stat].

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[57] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "TensorFlow: a system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[58] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[59] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, pp. 218–229, 2021.

[60] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[61] M. A. Nabian, R. J. Gladstone, and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," *Computer-Aided Civil and Infrastructure Engineering*, vol. 36, no. 8, pp. 962–977, 2021.

[62] P.-Y. Chuang and L. A. Barba, "Experience report of physics-informed neural networks in fluid simulations: Pitfalls and frustration," *Proceedings of the 21st Python in Science Conference*, pp. 28–36, 2022. Conference Name: Proceedings of the 21st Python in Science Conference.

[63] M. Tavelli and M. Dumbser, "A staggered semi-implicit discontinuous Galerkin method for the two dimensional incompressible Navier–Stokes equations," *Applied Mathematics and Computation*, vol. 248, pp. 70–92, Dec. 2014.

[64] M. Dumbser, I. Peshkov, E. Romenski, and O. Zanotti, "High order ADER schemes for a unified first order hyperbolic formulation of continuum mechanics: Viscous heat-conducting fluids and elastic solids," *Journal of Computational Physics*, vol. 314, pp. 824–862, 2016.

[65] A. Baïri, E. Zarco-Pernia, and J.-M. G. De María, "A review on natural convection in enclosures for engineering applications. The particular case of the parallelogrammic diode cavity," *Applied Thermal Engineering*, vol. 63, no. 1, pp. 304–322, 2014. Publisher: Elsevier.

[66] L. Q. Tang and T. T. Tsang, "A least-squares finite element method for time-dependent incompressible flows with thermal convection," *International Journal for Numerical Methods in Fluids*, vol. 17, no. 4, pp. 271–289, 1993. Publisher: Wiley Online Library.

[67] M. Z. Hossain, C. D. Cantwell, and S. J. Sherwin, "A spectral/hp element method for thermal convection," *International Journal for Numerical Methods in Fluids*, vol. 93, no. 7, pp. 2380–2395, 2021. Publisher: Wiley Online Library.

[68] A. Karakuş, "An accelerated nodal discontinuous Galerkin method for thermal convection on unstructured meshes: Formulation and validation," *Journal of Thermal Science and Technology*, pp. 91–100, 2022.

[69] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[70] K. Zubov, Z. McCarthy, Y. Ma, F. Calisto, V. Pagliarino, S. Azeglio, L. Bottero, E. Luján, V. Sulzer, A. Bharambe, N. Vinchhi, K. Balakrishnan, D. Upadhyay, and C. Rackauckas, "NeuralPDE: Automating physics-informed neural networks (PINNs) with error approximations," *arXiv:2107.09443*, 2021.

[71] C. Rao, H. Sun, and Y. Liu, "Physics-informed deep learning for incompressible laminar flows," *Theoretical and Applied Mechanics Letters*, vol. 10, no. 3, pp. 207–212, 2020.

[72] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: a review," *Acta Mechanica Sinica*, 2022.

[73] A. Karakus, N. Chalmers, K. Świrydowicz, and T. Warburton, "A GPU accelerated discontinuous Galerkin incompressible flow solver," *Journal of Computational Physics*, vol. 390, pp. 380–404, 2019.

[74] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, and S. Choudhry, "NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework," in *International Conference on Computational Science*, pp. 447–461, Springer, 2021.

[75] K. Stokos, S. Vrahliotis, T. Pappou, and S. Tsangaris, "Development and validation of an incompressible Navier-Stokes solver including convective heat transfer," *International Journal of Numerical Methods for Heat & Fluid Flow*, vol. 25, pp. 861–886, 2015.

[76] G. De Vahl Davis, "Natural convection of air in a square cavity: A bench mark numerical solution," *International Journal for Numerical Methods in Fluids*, vol. 3, no. 3, 1983.

[77] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018.

[78] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[79] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[80] R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter, "Multiplicative filter networks," in *International Conference on Learning Representations*, 2021.

[81] S. Habchi and S. Acharya, "Laminar mixed convection in a partially blocked, vertical channel," *International Journal of Heat and Mass Transfer*, vol. 29, no. 11, pp. 1711–1722, 1986.

[82] J. T. Batina, "Unsteady Euler airfoil solutions using unstructured dynamic meshes," *AIAA journal*, vol. 28, no. 8, pp. 1381–1388, 1990.

[83] B. A. Robinson, J. T. Batina, and H. T. Yang, "Aeroelastic analysis of wings using the Euler equations with a deforming mesh," *Journal of Aircraft*, vol. 28, no. 11, pp. 781–788, 1991.

[84] Y. Bazilevs, V. M. Calo, Y. Zhang, and T. Hughes, "Isogeometric fluid–structure interaction analysis with applications to arterial blood flow," *Computational Mechanics*, vol. 38, no. 4, pp. 310–322, 2006.

[85] K. Stein, R. Benney, V. Kalro, T. E. Tezduyar, J. Leonard, and M. Accorsi, "Parachute fluid–structure interactions: 3-d computation," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, no. 3-4, pp. 373–386, 2000.

[86] T. E. Tezduyar, S. Sathe, J. Pausewang, M. Schwaab, J. Christopher, and J. Crabtree, "Interface projection techniques for fluid–structure interaction modeling with moving-mesh methods," *Computational Mechanics*, vol. 43, no. 1, pp. 39–49, 2008.

[87] T. E. Tezduyar, M. Behr, S. Mittal, and J. Liou, "A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space-time procedure: II. computation of free-surface flows, two-liquid flows, and flows with drifting cylinders," *Computer methods in applied mechanics and engineering*, vol. 94, no. 3, pp. 353–371, 1992.

[88] T. E. Tezduyar, "Finite element methods for flow problems with moving boundaries and interfaces," *Archives of Computational Methods in Engineering*, vol. 8, no. 2, pp. 83–130, 2001.

[89] A. A. Johnson and T. E. Tezduyar, "Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces," *Computer methods in applied mechanics and engineering*, vol. 119, no. 1-2, pp. 73–94, 1994.

[90] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne, "Torsional springs for two-dimensional dynamic unstructured fluid meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 163, no. 1, pp. 231–245, 1998.

[91] K. Takizawa, T. E. Tezduyar, and R. Avsar, "A low-distortion mesh moving method based on fiber-reinforced hyperelasticity and optimized zero-stress state," *Computational Mechanics*, vol. 65, no. 6, pp. 1567–1591, 2020.

[92] R. Löhner and C. Yang, "Improved ALE mesh velocities for moving bodies," *Communications in numerical methods in engineering*, vol. 12, no. 10, pp. 599–608, 1996.

[93] I. Robertson and S. Sherwin, "Free-surface flow simulation using hp/spectral elements," *Journal of Computational Physics*, vol. 155, no. 1, pp. 26–53, 1999.

[94] B. T. Helenbrook, "Mesh deformation using the biharmonic operator," *International journal for numerical methods in engineering*, vol. 56, no. 7, pp. 1007–1021, 2003.

[95] S. Wang, S. Sankaran, and P. Perdikaris, "Respecting causality is all you need for training physics-informed neural networks," *arXiv preprint arXiv:2203.07404*, 2022.

[96] 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*, vol. 361, p. 112732, 2020.

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

[98] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *Neurocomputing*, vol. 317, pp. 28–41, 2018.

[99] E. Luke, E. Collins, and E. Blades, "A fast mesh deformation method using explicit interpolation," *Journal of Computational Physics*, vol. 231, no. 2, pp. 586–601, 2012.

[100] A. Shamanskiy and B. Simeon, "Mesh moving techniques in fluid-structure interaction: robustness, accumulated distortion and computational efficiency," *Computational Mechanics*, vol. 67, no. 2, pp. 583–600, 2021.

[101] A. de Boer, M. S. van der Schoot, and H. Bijl, "Mesh deformation based on radial basis function interpolation," *Computers & Structures*, vol. 85, no. 11, pp. 784–795, 2007.

[102] T. C. S. Rendall and C. B. Allen, "Efficient mesh motion using radial basis functions with data reduction algorithms," *Journal of Computational Physics*, vol. 228, no. 17, pp. 6231–6249, 2009.

[103] 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*, vol. 79, no. 11, pp. 1309–1331, 2009.