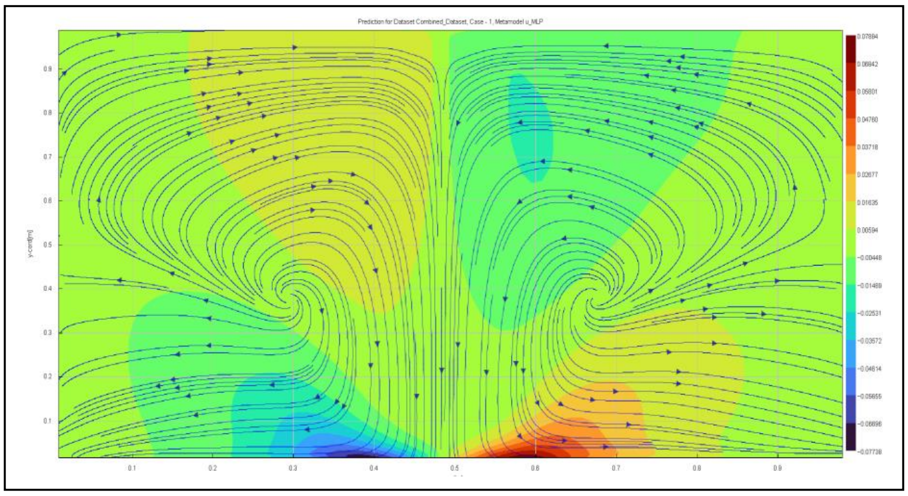
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr/

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr/

Master of Science Thesis

Physics-Informed Neural Networks for Data-Efficient & Accurate Learning of Physical Systems



Student: Panagiotis Koutsivitis
Registration Number: AIDL-0023

MSc Thesis Supervisor: Dr. Panagiotis Kasnesis

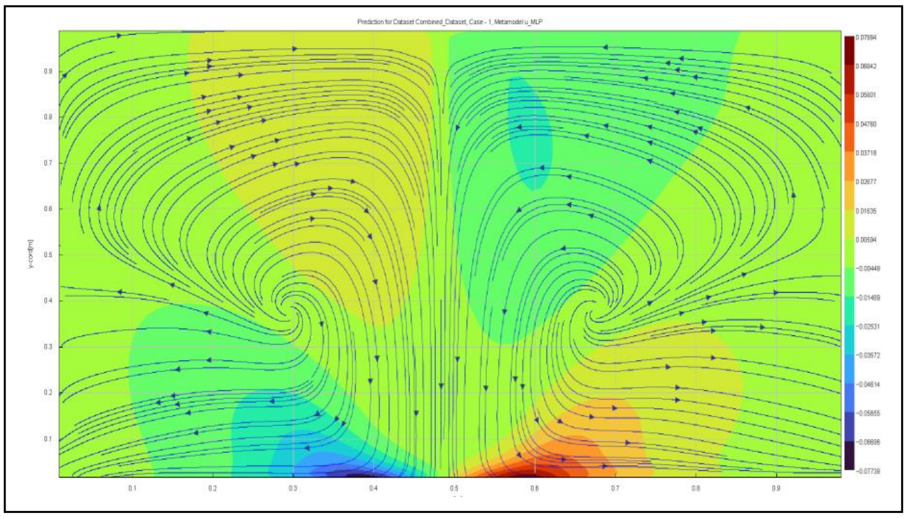
ATHENS-EGALEO, September 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΙ ΠΑΡΑΓΩΓΗΣ
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
Θηβών 250, Αθήνα-Αιγάλεω 12241
Τηλ: +30 210 538-1614
Διατμηματικό Πρόγραμμα Μεταπτυχιακών Σπουδών
Τεχνητή Νοημοσύνη και Βαθιά Μάθηση
https://aidl.uniwa.gr/

UNIVERSITY OF WEST ATTICA
FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
DEPARTMENT OF INDUSTRIAL DESIGN AND PRODUCTION ENGINEERING
http://www.eee.uniwa.gr
http://www.idpe.uniwa.gr
250, Thivon Str., Athens, GR-12241, Greece
Tel: +30 210 538-1614
Master of Science in
Artificial Intelligence and Deep Learning
https://aidl.uniwa.gr/

Μεταπτυχιακή Διπλωματική Εργασία

Νευρωνικά Δίκτυα υποβοηθούμενα από Φυσικούς Νόμους για την ακριβή (μηχανική) μάθηση φυσικών συστημάτων με αποδοτική χρήση συνόλων δεδομένων



Φοιτητής: Παναγιώτης Κουτσιβίτης
ΑΜ: AIDL-0023

Επιβλέπων : Δρ. Παναγιώτης Κασνέσης

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, Σεπτέμβριος 2024

This MSc Thesis has been accepted, evaluated and graded by the following committee:

Supervisor	Member	Member
PANAGIOTIS KASNESIS [illegible] Digitally signed by PANAGIOTIS KASNESIS Date: 2024.10.23 10:57:12 +03'00'	MARIA RAGKOUSI Digitally signed by MARIA RAGKOUSI Date: 2024.10.23 09:14:02 +03'00'	PERIKLIS PAPADOPOULOS Digitally signed by PERIKLIS PAPADOPOULOS Date: 2024.10.17 19:29:55 +03'00'
Kasnesis Panagiotis	Rangoussi, Maria	Papadopoulos, Pericles
Lecturer	Professor	Professor
Dept. of Electrical & Electronics Engineering	Dept. of Electrical & Electronics Engineering	Dept. of Electrical & Electronics Engineering
University of West Attica	University of West Attica	University of West Attica

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/η κάτωθι υπογεγραμμένος Παναγιώτης Κουτσιβίτης του Γεωργίου, με αριθμό μητρώου 0023 μεταπτυχιακός φοιτητής του ΔΠΜΣ «Τεχνητή Νοημοσύνη και Βαθιά Μάθηση» του Τμήματος Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών και του Τμήματος Μηχανικών Βιομηχανικής Σχεδίασης και Παραγωγής, της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Η εργασία δεν έχει κατατεθεί στο πλαίσιο των απαιτήσεων για τη λήψη άλλου τίτλου σπουδών ή επαγγελματικής πιστοποίησης πλην του παρόντος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.»

Ο Δηλών
Παναγιώτης Κουτσιβίτης

Παναγιώτης Κουτσιβίτης

(Υπογραφή φοιτητή)

Declaration of the author of this MSc thesis

I, Panagiotis Koutsivitis, George with the following student registration number: 0023, postgraduate student of the MSc programme in "Artificial Intelligence and Deep Learning", which is organized by the Department of Electrical and Electronic Engineering and the Department of Industrial Design and Production Engineering of the Faculty of Engineering of the University of West Attica, hereby declare that:
I am the author of this MSc thesis and any help I may have received is clearly mentioned in the thesis. Additionally, all the sources I have used (e.g., to extract data, ideas, words or phrases) are cited with full reference to the corresponding authors, the publishing house or the journal; this also applies to the Internet sources that I have used. I also confirm that I have personally written this thesis and the intellectual property rights belong to myself and to the University of West Attica. This work has not been submitted for any other degree or professional qualification except as specified in it.
Any violations of my academic responsibilities, as stated above, constitutes substantial reason for the cancellation of the conferred MSc degree.

The author
Panagiotis Koutsivitis

Panagiotis Koutsivitis

(Signature)

Στη μνήμη των γονιών μου, Γεώργιου και Μαργαρίτας.

In memory of my parents, George and Margarita

Acknowledgements

Above all, I extend my sincerest appreciation to my MSc thesis advisor, Dr. Panagiotis Kasnesis, for his guidance, support, and patience throughout my research journey. His insightful feedback, patience, and encouragement played a crucial role in the success of this project.
Finally, I would like to thank my colleagues Kai Jin and Navin Foglia at Gamma Technologies, for sharing their expertise in fluid dynamics and in general their invaluable insights and suggestions which have been instrumental in shaping this project.

Abstract

Traditionally, Scientific Computing and Computer Aided Engineering software use numerical solvers for simulating physics-based models. Despite their high accuracy, numerical solvers can become very computationally intensive and often are impractical to be applied in real time applications in Hardware-in-the-Loop systems (HiL), Electronic Control Units (ECU) or modern edge devices (Microcontrollers). On the other hand, many machine learning models such as Artificial Neural Networks are universal function approximators with very small inference, time and memory footprint. For the above reasons, modern scientific computing makes extensive use of Machine Learning for speeding up simulations or optimization processes. However, this use is limited by the presence of measurement or simulation data. In many cases, collecting measurements is not an option due to the high experimental costs, while in the case of simulation data, the need to run expensive in time simulations will often arise. Scientific Machine Learning or Physics-Informed Machine Learning tries to tackle the lack of training data by incorporating physics-based laws into the training process of machine learning models. More specifically, Physics Informed Neural Networks (PINNs) are a type of Neural Networks that are trained not only on data, if data are available, as it is usually the case in deep learning, but also on the model of the differential equations describing the underlying laws of physics, which makes them extremely accurate and data efficient. This ability to train a Neural Network in a non-arbitrary unsupervised way is a real breakthrough for the field of Machine Learning in general.

The aim of this MSc thesis is to apply the PINNs methodology in solving a variety of benchmark dynamical systems. We experimentally verify the ability of PINNs to solve standard benchmark problems such as Burgers equation and Poisson equation. We explore the borders of this technology by applying PINNs in challenging Fluid Dynamics problems, such as the Navier-Stokes equations. In Lid-Driven Cavity Flow, our trained PINNs demonstrate competitive performance in terms of accuracy when compared to established numerical solvers. Furthermore, they produce more precise results than those reported in a relevant PINN reference paper [1], all while utilizing significantly smaller neural network architectures. This becomes feasible by proposing and applying alternative optimization schemes. In the case of the Static Piston Flow problem, where PINNs failed to find the solution because of high nonlinearity and increased turbulence, we solve the problem using the classic Supervised Learning (SL) approach, which applies similar in size NN architectures, and we provide comparative results for the various optimizers used. The high accuracy achieved using the SL is a clear indication that the reason of PINNs' failure in that case was not the learning capability of the Neural Network but the complexity of the optimization problem itself. Finally, after extensive search in the bibliography, several candidate solutions have been gathered and are presented that could help to expand the limits of this technology and make it applicable to real world applications.

Keywords

Scientific Computing, Scientific Machine Learning, Deep Learning, Physics Informed Neural Networks, Automatic Differentiation.

Περίληψη

Παρά την εκτεταμένη χρήση της Μηχανικής Μάθησης στη σύγχρονη επιστημονική υπολογιστική, αυτή περιορίζεται από την παρουσία δεδομένων μέτρησης ή προσομοίωσης. Σε πολλές περιπτώσεις, η συλλογή δεδομένων μέσω καταγραφής και μετρήσεων δεν είναι εφικτή λόγω του υψηλού κόστους διεξαγωγής πειραμάτων, ενώ όσον αφορά τα δεδομένα προσομοίωσης, πολλές φορές απαιτείται η διεξαγωγή χρονοβόρων υπολογιστικών προσομοιώσεων. Η Επιστημονική Μηχανική Μάθηση, και πιο συγκεκριμένα το πεδίο Physics Informed Machine Learning, προσπαθεί να αντιμετωπίσει την έλλειψη δεδομένων εκπαίδευσης ενσωματώνοντας φυσικούς νόμους στην εκπαιδευτική διαδικασία των μοντέλων μηχανικής μάθησης. Πιο συγκεκριμένα, τα Physics Informed Neural Networks (PINNs) είναι μια κατηγορία Νευρωνικών Δικτύων που εκπαιδεύονται όχι μόνο σε δεδομένα, όπως συνηθίζεται στη βαθιά μάθηση, αλλά και στο θεμελιώδες μοντέλο των διαφορικών εξισώσεων που περιγράφει τους υποκείμενους φυσικούς νόμους – χαρακτηριστικό που τα καθιστά εξαιρετικά ακριβή και αποδοτικά ως προς τα δεδομένα.
Σκοπός της παρούσας μεταπτυχιακής διπλωματικής εργασίας είναι η εφαρμογή της μεθοδολογίας των PINNs για την επίλυση διάφορων δυναμικών συστημάτων αναφοράς. Επιβεβαιώνουμε πειραματικά την ικανότητα των PINNs να επιλύουν τυπικά προβλήματα αναφοράς, όπως η εξίσωση Burgers και η εξίσωση Poisson. Περαιτέρω, εξερευνούμε τα όρια της τεχνολογίας αυτής εφαρμόζοντας τη μέθοδο σε απαιτητικά δυναμικά προβλήματα Ρευστών, όπως τις εξισώσεις Navier-Stokes. Στο Lid-Driven Cavity Flow, τα εκπαιδευμένα PINNs μας επιδεικνύουν ανταγωνιστική απόδοση όσον αφορά την ακρίβεια, σε σύγκριση με καθιερωμένους αριθμητικούς επιλυτές. Επιπλέον, παράγουν πιο ακριβή αποτελέσματα από αυτά που αναφέρονται σε ένα σχετικό έγγραφο αναφοράς PINNs [1], ενώ όλα αυτά χρησιμοποιούν σημαντικά μικρότερες αρχιτεκτονικές νευρωνικών δικτύων. Αυτό γίνεται εφικτό με τη επιλογή και την εφαρμογή εναλλακτικών σχημάτων βελτιστοποίησης. Στην περίπτωση του προβλήματος Στατικής Ροής Εμβόλου, όπου τα PINN απέτυχαν να βρουν τη λύση λόγω της υψηλής μη γραμμικότητας και της αυξημένης τυρβώδους ροής, λύνουμε το πρόβλημα χρησιμοποιώντας την κλασική προσέγγιση εποπτευόμενης μάθησης (Supervised Learning, SL), εφαρμόζοντας αντίστοιχες σε μέγεθος αρχιτεκτονικές δικτύων, και παρέχουμε συγκριτικά αποτελέσματα για τους διάφορους βελτιστοποιητές που χρησιμοποιήθηκαν. Τέλος, μετά από εκτενή έρευνα στη βιβλιογραφία, έχουν συγκεντρωθεί και παρουσιάζονται αρκετές υποψήφιες λύσεις που θα μπορούσαν να βοηθήσουν στην επέκταση των ορίων αυτής της τεχνολογίας και να την καταστήσουν εφαρμόσιμη σε πραγματικές εφαρμογές.

Λέξεις – κλειδιά

Επιστημονική Υπολογιστική, Επιστημονική Μηχανική Μάθηση, Βαθιά Μάθηση, Νευρωνικά Δίκτυα με βάση τη Φυσική, Αυτόματη Παραγώγιση.

Table of Contents

List of figures

Acronym Index

CAE: Computer Aided Engineering

CFD: Computational Fluid Dynamis

FEM: Finite Element Method

ODE: Ordinary Differential Equation

PDE: Partial Differential Equation

AD: Automatic Differentiation

SL: Supervised Learning

NN: Neural Network

ANN: Artificial Neural Network

MLP: Multilayer Perceptron

RNN: Recurrent Neural Network

LSTM: Long Short-Term Memory

GRU: Gated Recurrent Unit

CNN: Convolutional Neural Network

GAN: Generative Adversarial Networks

PINN: Physics Informed Neural Network

NLP: Natural Language Processing

GD: Gradient Descent

SGD: Stochastic Gradient Descent

Adam: Adaptive Moment Estimation

BFGS: Broyden-Fletcher-Goldfarb-Shanno

L-BFGS: Limited Memory BFGS

LM: Levenberg-Marquardt

α-PINN: Automatic differentiation PINN

n-PINN: Numerical differentiation PINN

can-PINN: Coupled Automatic and Numerical differentiation PINN

XPINN: eXtended Physics-Informed Neural Networks

Re: Reynolds Number

MSE: Mean Squared Error

RMSE: Root Mean Squared Error

HiL: Hardware in the loop

ECU: Electronic Control Unit

Tanh: Hyperbolic Tangent

ReLU: Rectified Linear Function

MLA: Machine Learning Assistant

INTRODUCTION

Simulation of complex dynamical systems is a critical aspect of any industry that uses design software during product development. Scientific Computing and Computer Aided Engineering software use state-of-the-art numerical solvers for solving problems in - Fluid Dynamics, Thermal Management, Battery Modelling, Electromagnetics, Electro-chemical, multi-body dynamics and more. The process, while optimized, requires significant processing power that can take anywhere from several minutes to multiple days. This lengthy computation is often repeated numerous times due to variations in initial or boundary conditions and system parameters, leading to substantial redundant computational costs.

Traditional Neural Networks such as MLPs or other modern Deep Learning architectures, being universal function approximators, can learn simulation data that correspond to multiple simulation cases that vary in geometry, boundary conditions or system parameters. Such Neural Networks trained in diverse cases can be used to generate rapidly results even for unknown cases, which can be many orders of magnitude faster compared to the traditional simulation approaches, where transferring of results from one scenario to another is not feasible. However, these architectures, despite their learning ability and interpolation agility require the presence of data to learn from, which should be obtained by expensive experiments or time-consuming simulations. Moreover, there is not any theoretical guarantee that the trained Neural Networks respect the underlying physical laws of the problem.

The subject of this thesis

In 1997, Lagaris et al. [2] showed that even shallow Neural Networks that incorporate in their training process the differential equations of the system to be simulated, were able to solve the system achieving accurate results. Moreover, Raissi et al. [3] revisited Lagaris' method and introduced Physics Informed Neural Networks (PINNs) which take advantage of the modern deep neural network architectures and automatic differentiation techniques available in deep learning frameworks and so make feasible the solution of more challenging dynamical systems.

More specifically, PINNs is a generic method for solving a system of ordinary differential equations (ODEs) and partial differential equations (PDEs). Using the universal approximation theorem, the solution of such differential equations can be accurately estimated by a Neural Network. To train the parameters of the Neural Network, PINNs introduce a composite loss function that has multiple terms.

- **Initial/Boundary conditions Term**: This part of the loss function measures the difference between the network's predictions and the initial and/or boundary conditions data. It ensures that the neural network satisfies the initial and/or boundary conditions imposed by the problem. This is done by calculating the mean squared error (MSE) between the predicted values and the target values.
- **Physics Constraint Term**: This component quantifies the deviation of the network's predictions from the physical model described by the differential equations. It involves calculating the residuals of the differential equations at the input/collocation points and incorporating these residuals into the loss function. The goal is to minimize these residuals, indicating that the network's outputs comply with the physical laws.
- **Data Term (If data are present.)**: This part of the loss function measures the difference between the network's predictions and the actual data if any. It ensures that the neural network fits the known data points accurately. This is done by calculating the mean squared error (MSE) between the predicted values and the target values.

By minimizing this composite loss function, the network learns to predict outcomes that are not only consistent with the observed data, in the case that are available, but also aligned with the underlying physical principles. This dual-objective optimization ensures that the PINNs' predictions are physically plausible across a broader range of scenarios, beyond the specific instances represented in the training data.

Thesis' objectives

PINNs have gained significant attention as a promising method for solving various problems, such as computing ODEs or PDEs. The engineering community has started evaluating PINNs potential to replace, supplement or accelerate traditional approaches (numerical solvers) in various challenging tasks such as:

- solving linear or non-linear dynamical systems (forward problems)
- identifying system parameters (inverse problems)
- solving multi-case scenarios and creating fast surrogate models
- data assimilation
- uncertainty quantification

Objective of this thesis is to evaluate PINNs ability to solve well-known PDE benchmark problems in terms of accuracy and performance. These objectives will be pursued through a series of research questions, including:

- Can another optimizer outperform the common Adam + L-BFGS-based PINN optimization?
- How do PINNs perform to challenging multi-case dynamic fluid problems compared to data-driven approaches?
- What are the limitations and best practices of PINNs when it comes to solving complex PDEs?

The main contributions of this thesis

The main contributions of this thesis are summarized as follows:

- We implement a Python application using TensorFlow deep learning framework able to solve the following benchmark problems:
    - **Burgers equation**
    - **2-Dimensional Poisson equation**
- We solve the **Lid-Driven Cavity Flow** problem for various system parameter values. We diverge from the mainstream optimization approaches that use Adam + L-BFGS as main optimizers and we suggest the use of Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm (L-BFGS, a low memory version of BFGS, in particular). Our trained PINNs achieve a very competitive accuracy compared to state-of-the-art numerical solvers while giving more accurate results compared to a PINN reference paper [1], using significantly smaller neural network architectures.

- We stress test our solution and PINNs method in a challenging multi-case dynamic fluid problem that simulates the **flow inside a static piston** for different initial conditions. In this problem where the PINNs encountered difficulties due to the high nonlinearity and increased turbulence, we employed a traditional Supervised Learning (SL) approach to find a solution. We then compared the performance of various optimization algorithms within this framework. The high accuracy obtained through Supervised Learning clearly demonstrates that the limitations of PINNs were not related to the neural network's ability to learn complex patterns, but rather stemmed from the inherent complexity of the optimization problem itself.
- Following, we underline all the problems, weaknesses and pathologies found during these tests. Finally, after detailed research in the bibliography we present possible solutions to the weaknesses to make PINNs a more robust, effective and applicable method in engineering and beyond.

Structure

The rest of the thesis is organized as follows:

In Chapter 1, we will delve into the fundamental ideas and technological foundations that form the basis of PINNs. We'll explore how these concepts have developed over time and examine the current state of the Scientific Computing, differential equations (ODEs and PDEs), Neural Networks as universal function approximators, Automatic Differentiation, Optimization algorithms and PINNs.

Chapter 2 will present in detail all the benchmark problems will be solved using PINNs. This includes the specific theoretical background, differential equations and description of the domain and boundary conditions of the problems.

Chapter 3 will describe in detail the methodology followed for solving all the problems. The results of PINNs training are presented using advance visualizations. Additionally, comparisons are made between PINNs and Supervised Learning where a detailed dataset was available.

Chapter 4 aims to provide a comprehensive understanding of the limitations and challenges associated with PINNs, while at the same highlights many state-of-the-art improvements suggested in the literature to make PINNs more robust and effective to solving PDEs and complex physical systems.

Finally, the thesis concludes with a Conclusion section, where are summarized all the achievements, findings and comparative results, as well as certain suggestions for future work.

1 CHAPTER 1: Background

We will embark on an exploration of the core principles and technological building blocks that underpin PINNs. We will trace the evolution of these concepts over time and examine the present state of several key areas within Scientific Computing.

1.1 Scientific Computing

Scientific computing encompasses a broad spectrum of methodologies and technologies designed to solve complex in Science and Engineering through the application of computational mathematical models. Numerical analysis, a cornerstone of scientific computing, has its roots in mathematical concepts and techniques that predate the invention of electronic computers. These foundational principles, developed over centuries through continuous refinement, form the bedrock upon which modern scientific computing is built. The introduction of electronic computers was a pivotal moment in scientific problem-solving. This technological advancement necessitated a radical reassessment of established numerical methods, prompting widespread revision and, in some cases, complete overhaul of existing techniques. As electronic computers entered the scene, factors once considered trivial in manual calculations suddenly became paramount for optimal performance and accuracy in large-scale computations. This shift necessitated the establishment of a new academic discipline - Computer Science - which would encompass a wide range of critical components essential for effective scientific computing. However, Mathematics remains an indispensable cornerstone of scientific computing, serving multiple crucial functions in this interdisciplinary field. Its influence extends beyond problem formulation, encompassing key aspects of model validation, algorithmic development, and computational strategy. In summary, Scientific computing represents a synergistic fusion of mathematical principles and computational expertise in various Science Disciplines. This fusion can be seen in Figure 1.

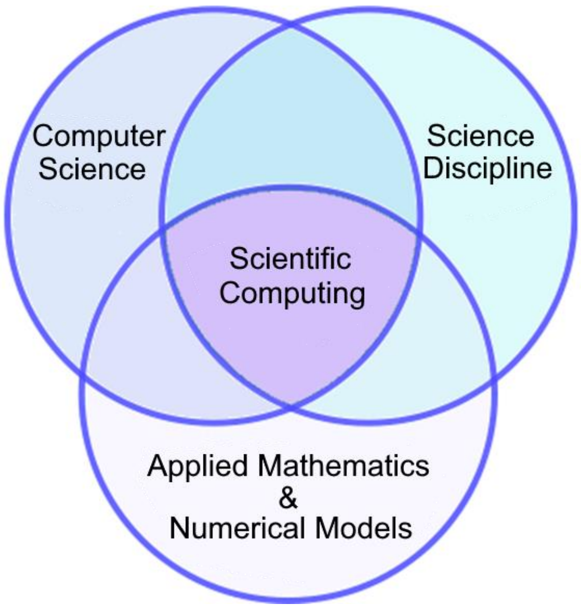


Figure 1. Scientific Computing [4]

1.1.1 Computer Aided Engineering (CAE)

Computer-aided engineering (CAE) has become an indispensable tool across various industries, particularly those relying on sophisticated design software. This innovative approach revolutionizes the product development process by leveraging digital technologies to streamline design, testing, and simulation phases.

Some of the most used simulation types in CAE are the following:

- Structural Analysis Simulations
- Computational Fluid Dynamics
- Multiphysics Simulations
- Design optimization

These simulation types often overlap or are combined in various ways depending on the specific product and industry. The choice of simulation type(s) depends on the nature of the product, the environmental conditions it will face, and the specific performance criteria that need to be met. Modern CAE tools often offer integrated multi-physics simulations, allowing engineers to analyze complex systems involving multiple physical phenomena within a single analysis framework.

1.1.1.1 Computational Fluid Dynamics (CFD)

Computational Fluid Dynamics (CFD) is a computational methodology that leverages advanced algorithms and numerical techniques to simulate and analyze fluid behavior in various engineering contexts. This sophisticated tool enables researchers and engineers to predict and understand complex fluid dynamics phenomena without the need for physical prototypes. CFD is based on the conservation laws of mass, momentum, and energy. These governing equations form the mathematical foundation upon which fluid simulations are built. CFD is utilized across a wide range of industries and engineering disciplines, including:

- **Aerospace and Defense:** One of the primary uses of CFD in aerospace is aerodynamics analysis. Engineers can simulate airflow around aircraft components such as wings, fuselages, and control surfaces. This allows them to:
    - Optimize airfoil shapes for better lift-to-drag ratios
    - Analyze drag reduction techniques
    - Study vortex flows and stall characteristics
    - Design more efficient wing shapes for various flight conditions

  By leveraging CFD, aerospace companies can significantly reduce wind tunnel testing requirements, saving time and resources while still achieving accurate aerodynamic performance predictions.

- **Automotive:** CFD has become an essential tool in the automotive industry, revolutionizing various aspects of vehicle design, development, and optimization. Here's how CFD is utilized in automotive applications:
    - Aerodynamics and Drag Reduction
    - Thermal Management
    - Fuel Efficiency and Emissions

- o Electric Vehicle Battery Performance

1.1.1.2 Finite Element Method (FEM)

The Finite Element Method (FEM) is a computational approach used extensively in engineering and scientific fields. This method breaks down intricate problems into manageable, discrete components known as finite elements. These simplified elements are then combined to form a comprehensive system of equations that accurately represents the original problem. FEM proves particularly valuable in scenarios involving complex geometries or systems where finding analytical solutions is challenging. By leveraging this technique, researchers and engineers can overcome difficulties associated with complex mathematical modeling and gain insights into various physical phenomena.

The key concept of FEM is:

- **Discretization**: Problem’s domain is divided into small elements, typically triangles or quadrilaterals in 2D or tetrahedra in 3D. This process is known as discretization.
- **Element Matrices**: For each element, a system of equations is formulated based on the governing differential equations of the problem. These equations are assembled into matrices, which describe the behavior of the element under various conditions.
- **Assembly Process**: The individual element matrices are combined to form a global matrix that represents the entire domain. This process considers the connectivity between elements.
- **Solving the Global System**: Once the global matrix is formed, it can be solved using standard numerical methods such as Gaussian elimination or iterative solvers such as the conjugate gradient method. The solution provides the values at discrete points within the domain.

FEM finds extensive applications across various fields:

- **Structural Analysis**: To analyze stresses, strains, and deflections in structures under load.
- **Heat Transfer**: For modeling heat distribution in materials or fluids.
- **Fluid Dynamics**: In simulating fluid flow and pressure distribution around objects.
- **Electromagnetics**: For analyzing electromagnetic fields and wave propagation.
- **Geophysics**: In seismic analysis and oil exploration.

Advantages of FEM

- **Flexibility**: Can handle complex geometries and material properties.
- **Accuracy**: By refining the mesh, the accuracy of the solution can be improved.
- **Robustness**: Can handle both linear and nonlinear problems, including those involving large deformations.

Disadvantages of FEM

- **Computational Cost**: As the complexity of the model increases, so does the computational cost.
- **Mesh Sensitivity**: Solution’s quality depends heavily on the detail of the mesh. Poorly shaped or irregular elements can lead to inaccurate results.

- **Complexity**: Setting up a FEM model requires a deep knowledge of the physical problem, and the numerical techniques involved.

In summary, the Finite Element Method is a powerful tool for solving complex engineering and scientific problems numerically. Its ability to handle complex geometries and material behaviors makes it indispensable in many areas of modern technology.

1.2 Differential Equations

Differential equations are mathematical equations that describe how quantities change over time or space. They are fundamental tools in many areas of science, engineering, economics, and mathematics itself. The term "differential" comes from the fact that these equations involve derivatives, which measure rates of change. There are wo fundamental categories within the realm of differential equations, the Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs).

1.2.1 Ordinary Differential Equations (ODEs)

Ordinary Differential Equations (ODEs) are a type of differential equations that deals with functions of a single variable. This variable typically represents time, although it could also represent other parameters. Unlike Partial Differential Equations (PDEs), which involve multiple independent variables, ODEs focus solely on one variable. This makes them simpler to solve compared to PDEs, though they still encompass a wide range of complexity and difficulty levels.

The main characteristics of ODEs are the following:

- **Single Independent Variable**: The primary characteristic of ODEs is that they involve only one independent variable, most commonly time ((t)), but it could also be another parameter relevant to the problem being modeled.
- **Order**: An ODE is defined by the highest order of its derivatives. For example, an equation containing the first derivative is a first-order ODE, one with the second derivative is a second-order ODE, and so on. There is no upper limit to the order of an ODE.
- **Types**:

  First Order: Incorporates the first derivative of the dependent variable.

  Second Order: Incorporates the second derivative of the dependent variable.

  Higher Order: Incorporates derivatives of higher orders.

1.2.2 Partial Differential Equations (PDEs)

Partial Differential Equations (PDEs) are a wide-ranging category of differential equations that incorporate linear or non-linear multivariable functions and their partial derivatives. Unlike Ordinary Differential Equations (ODEs), which address single variable functions, PDEs can describe phenomena involving spatial variations, making them indispensable in fields such as physics, engineering, and applied mathematics.

The main characteristics of PDEs are the following:

- **Multiple Independent Variables**: PDEs involve functions of multiple independent variables, typically representing both space and time. For example, in a heat conduction problem, the temperature might depend on both location (spatial coordinates) and time.
- **Partial Derivatives**: The term "partial" refers to the fact that the derivatives in PDEs are taken with respect to some of the variables while holding others constant. This distinguishes them from total (or full) derivatives found in ODEs.
- **Complexity**: PDEs can vary greatly in complexity, ranging from simple linear equations to highly nonlinear ones. Their solutions can exhibit a wide range of behaviors, from smooth and continuous to discontinuous and chaotic.

1.3 Artificial Neural Networks

Artificial Neural Networks are fundamental tools in machine learning, powering many state-of-the-art algorithms and applications across various scientific and engineering domains, including scientific computing, time-series forecasting, natural language processing, computer vision, robotics, and more.

A neural network is composed of interconnected units called neurons, arranged in layers. These neurons, inspired initially by the biological neurons (Figure 2), act as processing nodes within the network. They receive incoming signals, process this information through mathematical non-linear operations, called activation functions, and then generate output signals. This output can be propagated to other neurons within the network. The typical structure of a neural network includes:

1. Input Layer: Where initial data enters the network
2. Hidden Layers: Perform complex computations on the input data
3. Output Layer: Generates predictions or makes decisions based on the processed information

Within these layers, neurons communicate through weighted connections. These connections, represented by numerical weights, determine the strength of influence one neuron's output has on another neuron's input. Think of these weights as the "importance" of each connection in shaping the overall network behavior. During the training process, the network learns to adjust these weights based on examples provided in a training dataset. This adjustment allows the network to refine its understanding of patterns and relationships within the data. The learning process involves iterative adjustments to these weights and biases. As the network processes more data and receives feedback, it continually refines its internal representations of the world, gradually improving its ability to make accurate predictions or decisions.

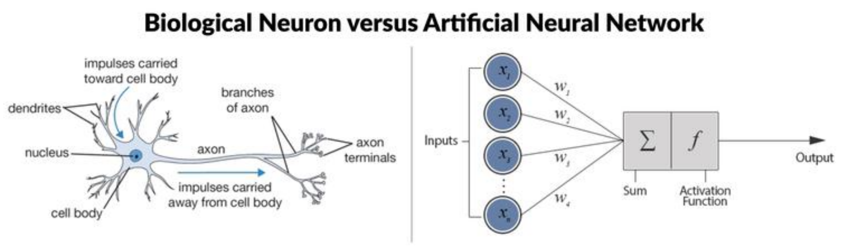


Figure 2. Biological Neuron versus Artificial Neural Network [5]

There are many kinds of Neural Networks such as:

Feedforward Neural Networks (FNN)

These are the most basic and straightforward form of Artificial Neural Networks (ANNs). In these networks, information flows in a linear, unidirectional manner, moving from the input layer to the output layer without any backtracking or circular paths. Multilayer perceptron (MLP) is a specific architecture of feedforward neural network.

Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network characterized by their unique architectural feature: recurrent connections. These connections create directed cycles within the network, enabling information to circulate and accumulate over time. This temporal memory capability makes RNNs particularly well-suited for tackling complex problems involving sequential or time-dependent data.

Convolutional Neural Networks (CNN)

CNNs are specialized neural networks engineered to efficiently process and analyze data that exhibits a grid-like structure, particularly images. These networks are built around a fundamental principle: the use of convolutional layers to extract meaningful features from the input data. They are composed of successive layers of convolutional filters that progressively build up hierarchical representations of features within the input data. CNNs are widely used in tasks such as image recognition, object detection, and image segmentation.

Long Short-Term Memory Networks (LSTM) and Gated Recurrent Units (GRU)

Long Short-Term Memory Networks (LSTMs) and Gated Recurrent Units (GRUs) represent advanced variants of recurrent neural networks (RNNs). These specialized architectures were developed to overcome the limitations of traditional RNNs, particularly the vanishing gradient problem. LSTMs and GRUs excel in processing sequential data with varying time scales.

Autoencoder

Autoencoders are unsupervised neural networks that excel at compressing and reconstructing data. They consist of an encoder network that condenses high-dimensional inputs into lower-dimensional latent representations, followed by a decoder that attempts to recreate the original data from these compressed forms. This process allows autoencoders to learn compact feature embeddings while discarding redundant information. They find applications in dimensionality reduction, anomaly detection, image denoising, and generative modeling. By learning hierarchical representations of data, autoencoders serve as powerful tools for exploratory data analysis and feature learning, enabling machines to identify and represent complex patterns within large datasets. Their unsupervised nature makes them particularly valuable for discovering hidden structures in unlabeled data, paving the way for various machine learning tasks and data preprocessing steps.

Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GANs) are a revolutionary deep learning framework consisting of two neural networks locked in a perpetual game of deception. The generator creates synthetic data samples that mimic the real thing, while the discriminator tries to spot the fakes. Through this adversarial dance, both networks continuously improve, with the generator becoming increasingly skilled at producing convincing forgeries and the discriminator developing superhuman abilities to detect authenticity. GANs have transformed the field of computer vision, effortlessly conjuring photorealistic images, videos, and even entire worlds from scratch. Their applications extend far beyond mere aesthetics, powering cutting-edge technologies in data augmentation, style transfer, and even creative endeavors like artistic collaborations between humans and AI. However, GANs are not without their challenges, as researchers grapple with issues like mode collapse and ensuring ethical use of these powerful generative models. Despite these hurdles, GANs remain at the forefront of AI innovation, pushing the boundaries of what's possible in data generation and manipulation.

Multilayer Perceptron

Multi-Layer Perceptrons (MLPs) are foundational neural network architectures that consist of multiple layers of interconnected nodes (neurons) processing information in a feedforward manner. Characterized by their layered structure (Figure 3), MLPs sequentially apply transformations to the input data, with each layer building upon the previous one to create increasingly complex representations. The MLP consists of an input layer, one or more hidden layers, and an output layer. MLPs are versatile tools capable of solving a wide range of problems, including classification, regression, and clustering tasks. Their layered structure allows them to learn hierarchical features, making them effective for pattern recognition and decision-making. While simpler than some modern neural network architectures, MLPs remain powerful and widely used, especially in scenarios requiring interpretable models or when computational resources are limited. Their linear flow of information and ease of implementation make MLPs accessible entry points for exploring neural network concepts and applying them to various machine learning challenges. Under certain mathematical conditions, MLPs possess the remarkable ability to approximate any function with arbitrary precision. This property makes them an indispensable tool in the field of artificial intelligence, providing a foundation upon which more advanced neural network architectures are built. The combination of their flexibility, computational efficiency, and theoretical robustness has cemented MLPs as a fundamental component in the development of deep learning models and ongoing research in neural network theory.

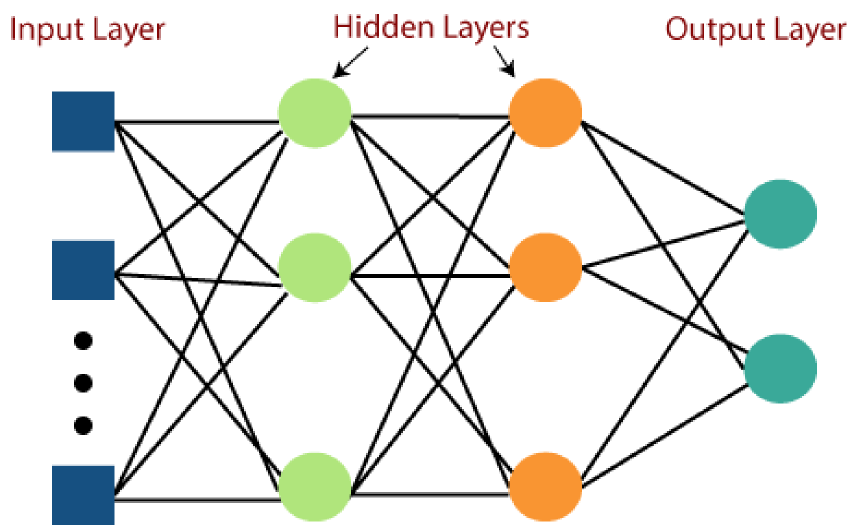


Figure 3. Abstract diagram of an MLP [6]

1.4 Computational Graphs

A computational graph, also known as a data flow graph or a dependency graph, is a graphical representation of computations performed during the execution of a program or algorithm. It visually maps out how data flows through an algorithm or system, showing dependencies between operations and the sequence in which they occur. Computational graphs are widely used in various fields such as computer science, mathematics, and engineering, particularly in areas like machine learning, optimization problems, and digital signal processing.

The key components of a Computational Graph are:

- **Nodes**: Represent individual operations or functions within the computation. Each node performs a specific task, such as arithmetic operations (addition, multiplication), activation functions in neural networks, or any other type of operation relevant to the problem being solved.
- **Edges**: Connect nodes and represent the flow of data between them. The direction of an edge indicates the order in which operations should be executed, ensuring that all necessary inputs are available when an operation is performed.
- **Data Flow**: Shows how information moves from one part of the graph to another (Figure 4). In some cases, the graph might have feedback loops where the output of a later stage feeds back into an earlier stage, allowing for iterative processes.

The main applications of Computational Graphs are:

- **Machine Learning**: Neural networks, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are frequently depicted and analyzed using computational graphs. These graphs help in understanding the architecture of the network, including layers, connections, and the flow of data.
- **Optimization Problems**: Many optimization algorithms, such as gradient descent, can be visualized using computational graphs. This visualization aids in understanding the steps involved in finding the optimal solution.
- **Algorithm Design and Analysis**: Computational graphs can help in analyzing the complexity of algorithms by visualizing the number of operations and their dependencies.

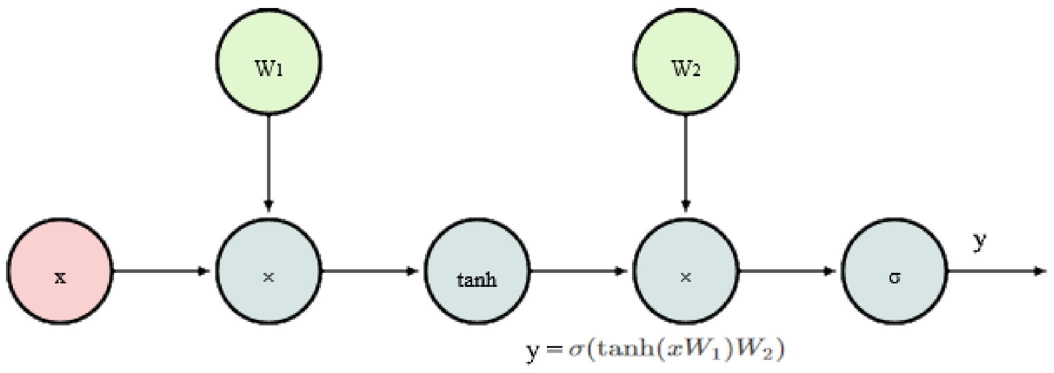


Figure 4. **Computational Graph used for calculating the forward pass of $y=\sigma(\tanh(xW_1)W_2)$ [7]**

1.5 Automatic Differentiation

Automatic Differentiation (AD) is a powerful technique that enables efficient computation of derivatives for complex functions. It leverages the fact that all computer calculations can be represented as a sequence of elementary operations and functions. By analyzing these operations within a computational graph, AD applies the chain rule repeatedly (Figure 5) to compute partial derivatives automatically. This process yields accurate results up to working precision, requiring only a small constant factor more arithmetic operations than the original program. AD's efficiency and accuracy make it an invaluable tool in machine learning and optimization, particularly for training deep neural networks. Its ability to handle intricate functions seamlessly has revolutionized gradient-based methods, enabling rapid development and deployment of sophisticated models across various domains.

Unlike numerical differentiation, which introduces round-off errors and cancels out terms, AD provides exact derivatives up to floating-point precision. AD excels at computing higher-order derivatives and partial derivatives with respect to many inputs, crucial for gradient-based optimization. This method solves the problems inherent in classical differentiation techniques, offering efficient and accurate computation of gradients. AD's power lies in its ability to handle complex functions seamlessly, making it an invaluable tool in machine learning and optimization, particularly for training deep neural networks.

The main Applications of AD are the following:

- **Optimization**: AD is crucial in optimization algorithms like any variant of gradient descent, which rely on gradients to update model parameters iteratively.
- **Machine Learning**: In deep learning, AD is used to train neural networks by optimizing the weights based on the gradients of the loss function with respect to these weights.
- **Scientific Computing**: AD can accelerate the computation of derivatives in simulations and experiments, enabling more accurate and efficient modeling.

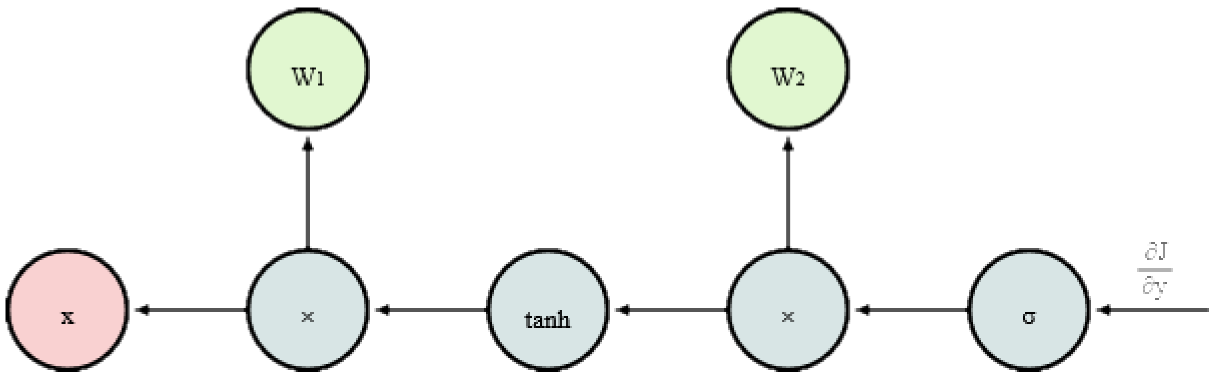


Figure 5. Computational Graph used for calculating the reverse pass of $y=\sigma(\tanh(xW_1)W_2)$ [7]

1.6 Optimization algorithms

Optimization is a broad concept that can be applied across various fields, including mathematics, computer science, engineering, and more. At its core, optimization involves finding the best solution among a set of possible solutions to achieve a specific goal or objective. In Scientific computing and CAE, optimization is used for model design optimization or model calibration. During the years have been developed many optimization algorithms that belong to different categories and can be applied to solve different types of problems. In this thesis we present briefly some of the most important categories of optimization algorithms.

1.6.1 Heuristic algorithms

The number of the possible solutions in an optimization problem depends on the number of variables that form the problem space and whether these variables are continuous or discrete (Integer Programming). The fact that this number can be very large or even infinite, finding the exact solution is impractical due to time constraints. Heuristic optimization algorithms are designed to solve complex problems by providing good-enough solutions within reasonable time frames, rather than always finding the absolute best solution. These algorithms are particularly useful in scenarios where the problem space is too vast, or the computation required to find the optimal solution is prohibitively expensive.

1.6.2 Genetic algorithms

Genetic algorithms are evolutionary computation techniques inspired by Darwin's theory of natural selection and genetics. They operate on a population of candidate solutions, applying principles of evolution to iteratively improve the solution set. The algorithm starts with an initial population of potential solutions, representing possible answers to a problem. It then applies genetic operators like selection, crossover, and mutation to these solutions, mimicking biological processes. Through repeated iterations, fitter solutions (those better addressing the problem) become more prevalent in the population. This evolutionary process allows the algorithm to explore vast solution spaces efficiently, often finding optimal or near-optimal solutions to complex problems. Genetic algorithms have been applied successfully in various fields, including optimization, scheduling, machine learning, and engineering design. Their ability to handle non-linear relationships and parallel processing makes them particularly effective for solving real-world problems with multiple constraints and objectives.

1.6.3 Gradient Descent

Gradient Descent (GD) is a deterministic optimization algorithm and belong specifically to the gradient-based optimization methods. GD is one of the simplest and most widely used first-order optimizers. It iteratively updates parameters in the direction opposite to the gradient of the loss function.

1.6.3.1 1st Order Optimizers

First-order optimizers are a fundamental class of optimization algorithms commonly used in machine learning and deep learning. These optimizers rely solely on the 1st order gradient information of the loss function (Figure 6) to update model parameters.

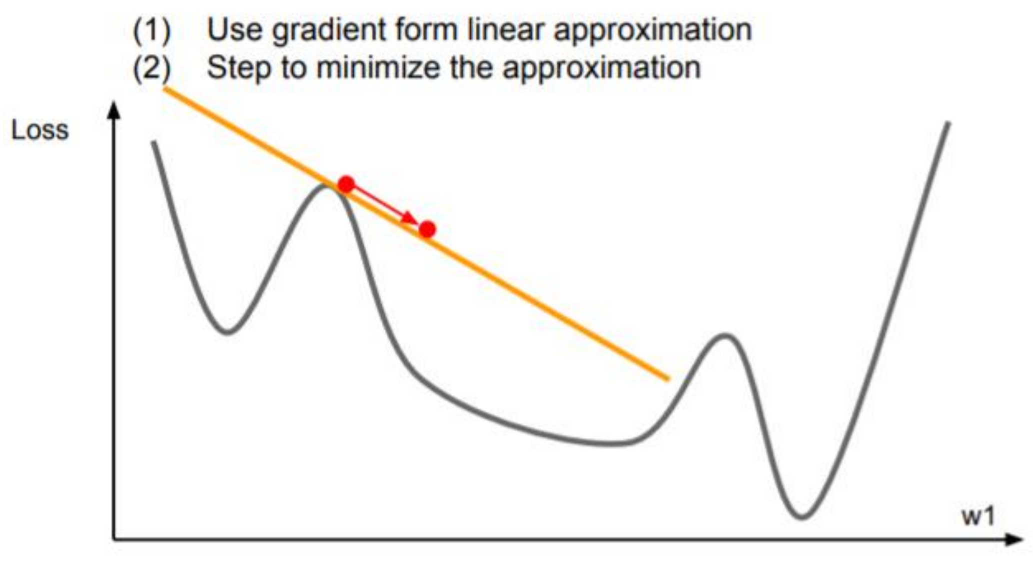

(1) Use gradient form linear approximation
(2) Step to minimize the approximation
Loss
w1

Figure 6. Gradient Descent using 1st order optimizers [8]

Some of the advantages and disadvantages of the 1st order optimizers are the following:

- Pros:
    - ➢ Low computational cost per epoch
    - ➢ Scalable to any number of parameters
    - ➢ Perform well in stochastic objective functions.
- Cons
    - ➢ Slow convergence due to the small steps taken
    - ➢ Sensitive to hyper-parameters
    - ➢ Need more parameters for finding accurate solutions than it should.

1.6.3.1.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is a variant of Gradient Descent that uses only one or a limited number of examples from the training dataset at a time to compute the gradient. This approach contrasts with traditional Gradient Descent, which computes the gradient using the entire dataset. This stochastic approach to the calculation of gradient has been found to have the following advantages compared to the GD:

- Faster computation compared to full batch GD
- Noisier updates, leading to better generalization
- Often converges to a good solution quickly

1.6.3.1.2 Adam

The Adam algorithm [9] is a first-order stochastic gradient-based optimizer that has gained popularity for its effectiveness in training large and complex deep neural networks. This method stands out for its robustness and computational efficiency, making it particularly well-suited for models with millions of parameters. Adam's success in deep learning can be attributed to its ability to adapt learning rates for different parameters automatically, reducing the need for manual tuning. However, achieving optimal performance often requires careful consideration of several hyper-parameters which in practice can be challenging and time consuming.

1.6.3.2 2nd Order Optimizers

Second-order optimizers in machine learning refer to optimization algorithms that utilize both the first and second derivatives of the loss function (Figure 7) during the training process. These optimizers aim to converge faster and more efficiently compared to first-order methods, which only rely on the gradient (first derivative) information.

1. Utilize Hessian matrix: Second-order optimizers make use of the Hessian matrix, which contains the second partial derivatives of the loss function.
2. Faster convergence: By incorporating curvature information from the Hessian matrix, second-order methods often converge faster than first-order methods.
3. Computational complexity: Second-order optimizers typically require more computational resources due to the need to compute and invert the Hessian matrix.

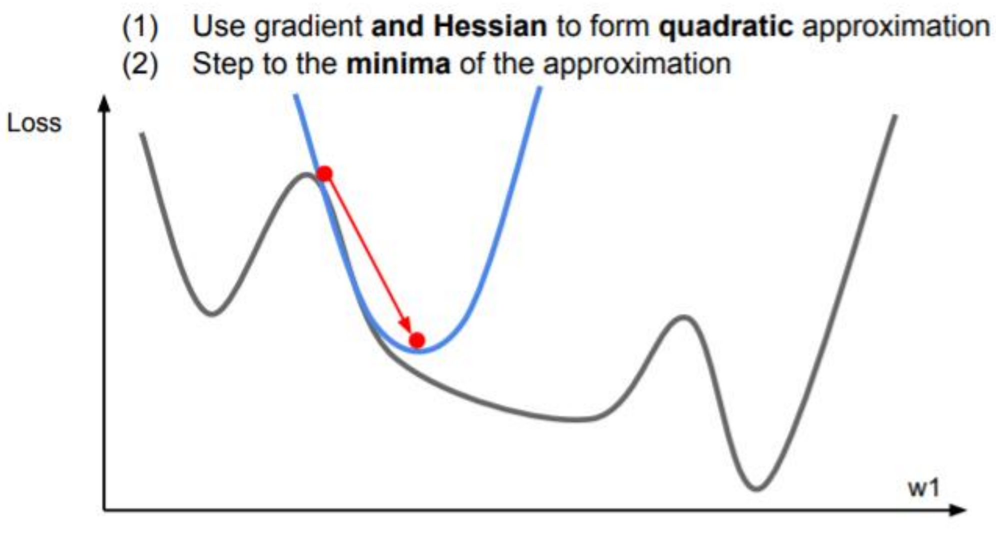

(1) Use gradient **and Hessian** to form **quadratic** approximation
(2) Step to the **minima** of the approximation
Loss
w1

Figure 7. Gradient Descent using 2nd order optimizers [8]

Some of the advantages and disadvantages of the 2nd order optimizers are the following:

- Pros:
    - ➢ By incorporating curvature information from the Hessian matrix, 2nd order methods often converge faster than 1st order methods.
    - ➢ Not at all or few hyper-parameters

- Need less parameters for finding accurate solutions than 1st order optimizers.

- Cons
  - High computational cost per epoch because of the required estimation & inversion of the Hessian matrix.
  - High memory consumption because of the allocation of the Hessian matrix.
  - Impractical for large number of parameters.
  - Do not perform well in stochastic objective functions.

1.6.3.2.1 BFGS

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is a second-order optimization method that approximates the Hessian matrix of the loss function. It achieves this by evaluating gradients, resulting in a computational complexity of $O(N^2)$ where N is the number of parameters. A significant advantage of BFGS is that it updates the curvature matrix without requiring matrix inversion, significantly reducing computational costs. However, this method comes with memory constraints, as the Hessian matrix grows quadratically with the number of parameters. This limitation makes BFGS unsuitable for large-scale neural networks, where memory usage becomes prohibitively expensive as the model size increases. Despite these challenges, BFGS remains valuable in certain scenarios due to its ability to handle complex landscapes efficiently. Its effectiveness in finding optimal solutions has led to continued research into variants and applications of the algorithm in various optimization problems beyond deep learning.

1.6.3.2.2 L-BFGS

The Limited-memory BFGS (L-BFGS) algorithm [10] is an improved version of the BFGS algorithm that addresses memory usage concerns by storing only a few vectors approximating the full Hessian matrix. This approach enhances computational efficiency and allows L-BFGS to handle problems with larger parameter sets compared to BFGS.

Both BFGS and L-BFGS can still fall prey to local minima traps. To mitigate this risk, researchers have developed hybrid optimization strategies. One such approach involves using Adam as an initial optimizer, running for hundreds of epochs, followed by BFGS or L-BFGS using the weights produced by Adam. This multi-stage strategy aims to leverage Adam's global exploration capabilities with the local refinement offered by BFGS variants. This combined approach has shown promise in physics-informed machine learning applications, particularly in solving systems of partial differential equations (PDEs) [3]. By combining these methods, researchers can potentially achieve better convergence and avoid getting stuck in suboptimal solutions, especially in challenging optimization landscapes encountered in complex scientific computing tasks.

1.6.3.2.3 Levenberg-Marquardt

The Levenberg-Marquardt (LM) method originated from Levenberg's work on non-linear least squares optimization in 1944. He observed that gradient descent and Gauss-Newton iteration were complementary approaches. Marquardt later extended this method in 1963 by incorporating the Hessian matrix's diagonal, scaling each component of the gradient according to curvature. This extension, now known as the Levenberg-Marquardt algorithm, combines the strengths of both methods. While LM is generally extremely effective for moderate-sized models, it has limitations due to its requirement for matrix inversion, which is computationally expensive and memory-intensive. This constraint traditionally restricted LM to models with thousands of parameters. However, advancements in modern computing hardware have significantly increased the size of models that can benefit from LM optimization. Today, LM remains a valuable tool in the optimization toolkit, particularly for problems where a good balance between fast convergence and robustness is desired. Its ability to handle non-linear relationships makes it applicable to various real-world optimization problems, though care must still be taken regarding model size and computational resources.

1.7 PINNs

In many cases, a system of differential equations can be described by

$D[u(x); \lambda] = f(x)$, $x \in \Omega$,
$B_k[u(x)] = g_k(x)$, $x \in \Gamma_k \subset \partial\Omega$,
for $k = (1, 2 \ldots, n_b)$

D is a differential operator, B_k is a set of boundary operators, $u \in R^{du}$ is the solution to the differential equation(s), f (x) is a forcing function, $g_k(x)$ is a set of boundary functions, x is an input vector in the domain $\Omega \subset Rd$ (i.e. x is a d-dimensional vector), $\partial\Omega$ denotes the boundary of Ω and λ is a set of additional parameters of the differential operator.
A PINN is a neural network, NN(x; θ), with trainable weights and biases θ that aims to approximate the exact solution u(x) of the underlying differential equation, i.e. $NN(x; \theta) \approx u(x)$. To train the PINN, a multi-component loss function is employed, consisting of at least two terms or more when observational data is present (Figure 8).

$L(\theta) = \alpha_1 L_{bound}(\theta) + \alpha_2 L_{physics}(\theta) + \alpha_3 L_{data}(\theta)$

- The term $L_{bound}(\theta)$ is the "boundary" loss which tries to match the PINN solution to the known solution along the boundaries of the domain and typically is calculated using the following formula $L_{bound}(\theta) = \Sigma\, (u(x) - NN(x; \theta))^2$
- The term $L_{physics}(\theta)$ is the "physics" loss which tries to minimize the residual of the underlying equation(s) at a set of locations within the domain. The derivatives of the PINN solution with respect to its inputs required by the boundary and physics loss are obtained using automatic differentiation.
- The term $L_{data}(\theta)$ is the "data" loss which tries to match the PINN solution to measured known solutions. Usually, these solutions are located within the domain of the problem. The $L_{data}(\theta)$ is usually used when an inverse problem should be solved.

The hyper-parameters $\alpha_1, \alpha_2, \alpha_3$ control the influence of each loss term in the optimization problem and can play a significant role in the convergence time and results accuracy.

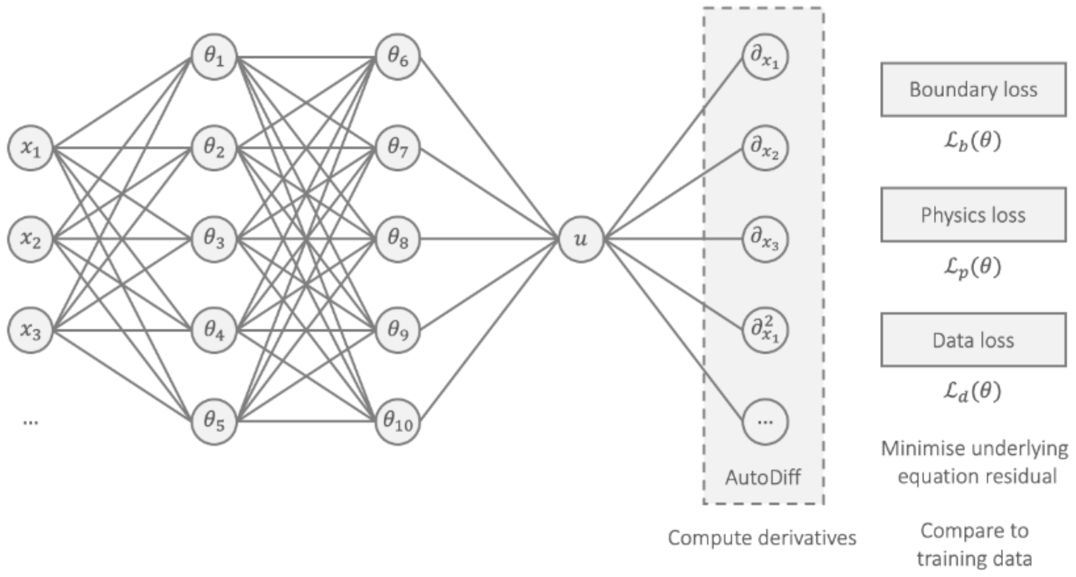


Figure 8. Illustration of how the loss function of PINNs is formed using the computed derivatives of the NN by Automatic Differentiation and boundary & data loss terms [11]

1.7.1 Forward

Traditionally, given a system of differential equations, simulation software is used for solving forward problems where you have a set of initial or boundary conditions and want to predict the state of the system at future times or under different conditions. From the very beginning, PINNs methodology has been used for solving this type of problems with a good success. Despite their success, PINNs have been found to meet several challenges because of:

- Large training times compared to traditional numerical approaches.
- Poor accuracy in highly non-linear problems and large domains.
- Difficulty in generalization to new problem cases.

However, PINNs still offer some advantages that is very difficult to find in the traditional solvers:

- are Mesh free, which means that do not rely necessarily on traditional numerical discretization methods that require a structured grid or mesh and so can be:
    1. **Efficient in high-dimensional problems**: Traditional methods often suffer from the curse of dimensionality, where the computational cost grows exponentially with the number of dimensions. Mesh-free methods like PINNs can handle high-dimensional problems more efficiently because they do not require the explicit construction of a mesh.
    2. **Flexible in Complex Geometries**: PINNs can naturally accommodate complex geometries without needing to adapt or refine meshes. This flexibility is crucial in applications involving irregular domains or interfaces.
    3. **Parallelizable**: The structure of PINNs lends itself well to parallel computing architectures, allowing for efficient scaling across multiple processors or GPUs.

- are tolerant to the low-quality or absence of initial or boundary conditions when experimental data are provided and integrated into the training process alongside with the differential equations, even if the provided data are corrupted by noise.

1.7.2 Inverse

Inverse problems play a vital role in numerous real-world applications. Here the goal is to estimate the parameters of a system given a set of real-world observations of the system that potentially can be corrupted by noise (Figure 9). Inversion can be challenging computationally wise. Solving inverse problems using black-box optimizations algorithms can be extremely demanding due to the need for extensive forward simulations to match model predictions with observations. This approach can become prohibitively expensive, especially for high-dimensional complex systems, rendering many applications infeasible. On the contrary, PINNs can be a viable solution for solving challenging inverse problems since with one single training, can estimate the system parameters of the underlying differential equations alongside with the optimized θ neural network parameters.

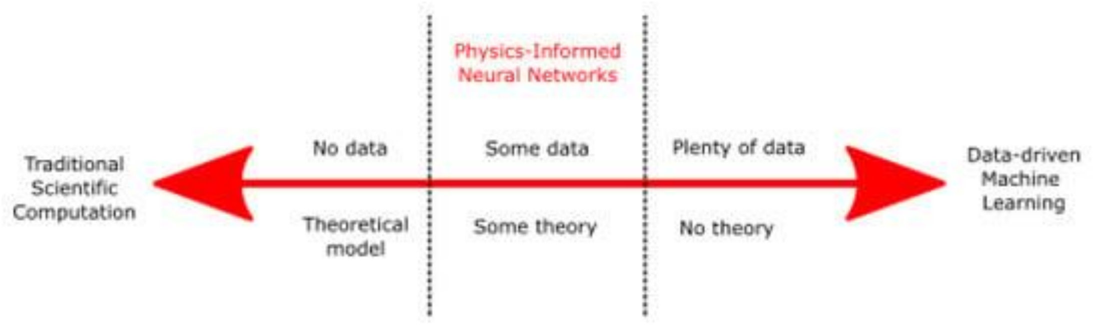

Physics-Informed Neural Networks
Traditional Scientific Computation
No data
Some data
Plenty of data
Data-driven Machine Learning
Theoretical model
Some theory
No theory

Figure 9. **Illustration of when PINNs can be used based on how many data of theory are available [12]**

2 CHAPTER 2: Experiments

In this chapter will be presented the theoretical background about the problems to be solved using PINNs. Even though PINNs have been found to be most effective in solving inverse problems, in this thesis all the problems to be solved belong to the class of forward problems. The reason behind this decision is to explore the capabilities of this innovative technology and examine whether could be competitive to the established traditional numerical methods in several benchmark problems.

2.1 Burgers equation

Burgers equation emerges across numerous fields within applied mathematics such as fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow. This equation serves as a foundational partial differential equation, which can be obtained by simplifying the Navier-Stokes equations related to velocity fields through the omission of the pressure gradient term.

When dealing with low viscosity parameters, Burgers equation may result in shock waves that pose significant challenges for resolution via traditional numerical approaches. Specifically, in a one-dimensional space setting, Burgers equation, accompanied by Dirichlet boundary conditions, is expressed as follows.

$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - v\frac{\partial^2 u}{\partial x^2} = 0, \qquad x \in [-1,1], \qquad t \in [0,1] \\ &u(0,x) = -sin(\pi x), \\ &u(t,-1) = u(t,1) = 0, \end{aligned} \qquad \textbf{Eq. 1}$$

Where v is the kinematic viscosity parameter and for our case will be equal to 0.01/π.

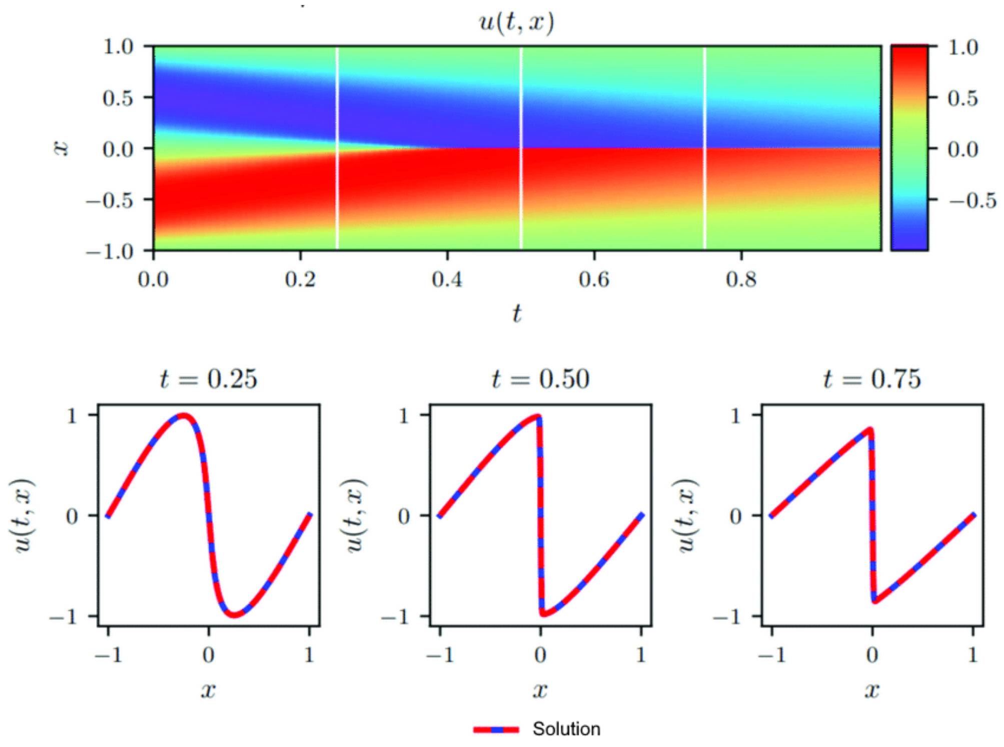


Figure 10. **Illustration of the domain and state space of Burgers equation [3]**

In Figure 10 it is illustrated the solution where can be seen the shock waves (discontinuities) generated as the time increases.

2.2 Poisson equation

The Poisson differential equation is a partial differential equation that arises in many areas of physics and engineering. It is named after Simeon Denis Poisson, a French mathematician who made significant contributions to the field of mathematical physics.

The importance of the Poisson differential equation lies in its wide range of applications across various fields. Its versatility stems from its ability to model the spatial distribution of physical quantities under the influence of sources or sinks, thereby providing a mathematical framework for understanding complex natural processes. It describes phenomena related to electrostatics, heat conduction, fluid dynamics, and many other areas. Here are some reasons why it is important:

- Electrostatics: It helps in calculating electric potentials given charge distributions.
- Heat Conduction: It models how heat diffuses through materials.

- Fluid Dynamics: It can describe the velocity potential in an irrotational flow.
- Gravitational Fields: It is used to calculate gravitational potentials and fields.

Without loss of generality in this example we are going to solve the Poisson equation in a unit square domain and Dirichlet boundary conditions described as follows:

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} = \frac{1}{4}\sum_{k=1}^{4}(-1)^{k+1}\, 2k\sin(k\pi x)\sin(k\pi y),$$
$$x \in [0,1],$$
$$y \in [0,1],$$
$$u(0,y) = 0, u(1,y) = 0, u(x,0) = 0, u(x,1) = 0 \quad \textbf{Eq. 2}$$

The specific formulation of the problem together with the source term in the right-hand side of the equation found in the scientific paper “Physics-Informed Deep-Learning for Scientific Computing” [13] by Stefano Markidis.

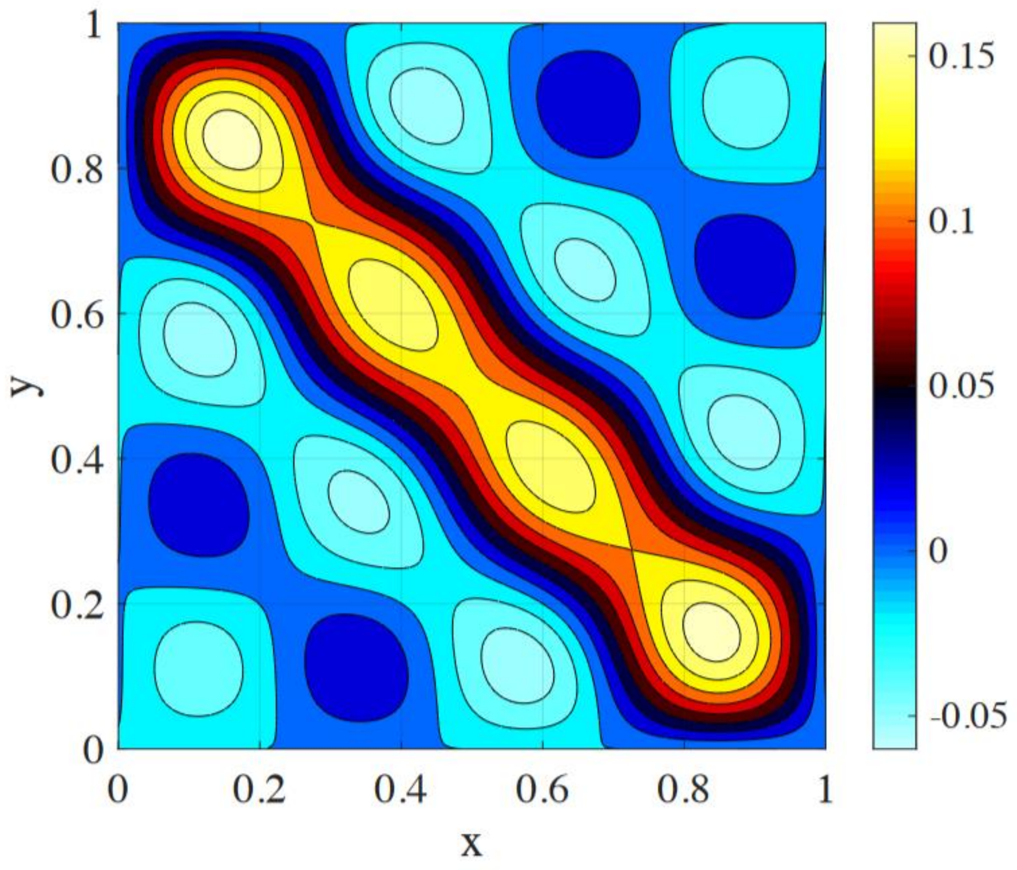


Figure 11. **Illustration of the domain and state space of the Poisson equation for the specific source terms [13]**

In Figure 11 can be seen the solution in the form of a 2-dimensional contour plot as has been found in the aforementioned research paper.

2.3 Lid Driven Cavity Flow Problem

The lid-driven cavity flow represents a classic problem in fluid dynamics that serves as a benchmark for testing numerical methods used to solve viscous, incompressible fluid flows.

This problem involves a square domain where three sides are stationary walls, while the top wall (the "lid") moves at a constant velocity U_{lid}. The movement of the lid induces a complex flow pattern within the cavity, characterized by a large primary vortex in the center and smaller secondary vortices near the corners (Figure 12).

This phenomenon is significant because it encapsulates several fundamental aspects of fluid dynamics, including boundary layer effects, vortex formation, and the influence of viscosity on flow patterns. The behavior of the fluid within the cavity is highly dependent on the Reynolds number, which quantifies the ratio of inertial forces to viscous forces within the fluid. At low Reynolds numbers, the flow is laminar and predictable, but as the Reynolds number increases, the flow becomes turbulent, leading to more complex and chaotic patterns.

The lid-driven cavity problem is particularly valuable because it provides a controlled environment for studying these effects. By varying parameters such as the lid velocity and the fluid viscosity, researchers can explore how changes in Reynolds number affect the flow dynamics. This makes it an excellent test case for validating computational fluid dynamics (CFD) algorithms and turbulence models.

Moreover, understanding the lid-driven cavity flow has practical applications in engineering and industrial processes. Many real-world fluid flow problems involve moving boundaries or surfaces that induce flow patterns like those observed in the cavity problem. Examples include the mixing processes in chemical reactors, the cooling systems in electronic devices, and the aerodynamics of vehicles. By studying the lid-driven cavity flow, engineers can gain insights into these complex flow phenomena, which can lead to more efficient designs and improved performance in various applications.

In summary, the lid-driven cavity flow problem is a cornerstone in fluid dynamics research due to its simplicity yet richness in demonstrating fundamental principles of viscous flows. It serves both as a theoretical benchmark for testing numerical methods and as a practical model for understanding and optimizing real-world engineering applications involving fluid flow.

For the lid-driven cavity flow, we consider a 1.0 × 1.0 m2 square domain, with $Ulid$ = 1 m/s, and Re = 100, 400, and 1000.

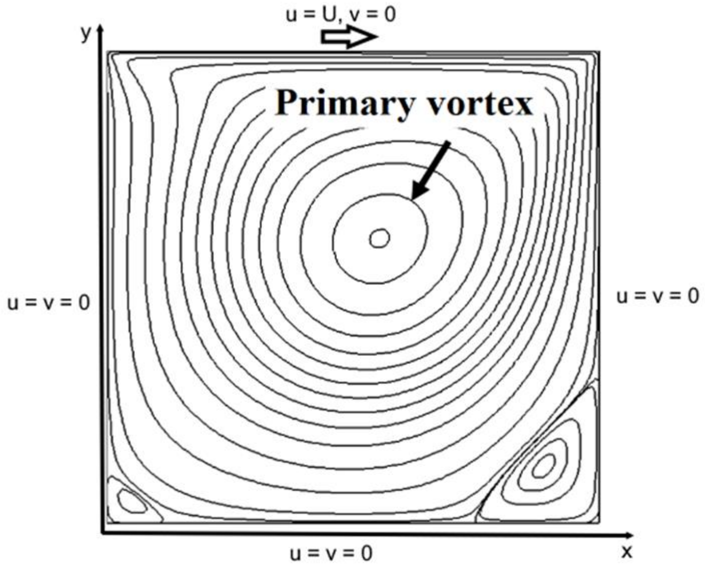
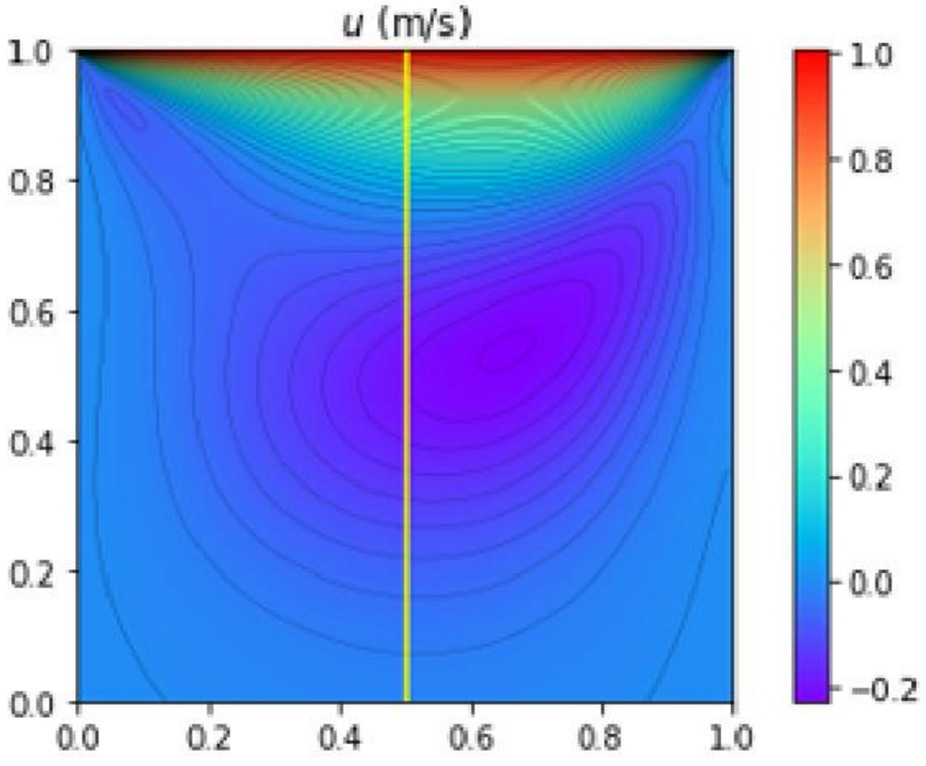


Figure 12. **Lid Driven Cavity Flow for Re=100, Illustration of u & v velocity field using streamlines. Can be seen the primary vortex on the center and the two secondary vortices on the left & right bottom corners. [14]**



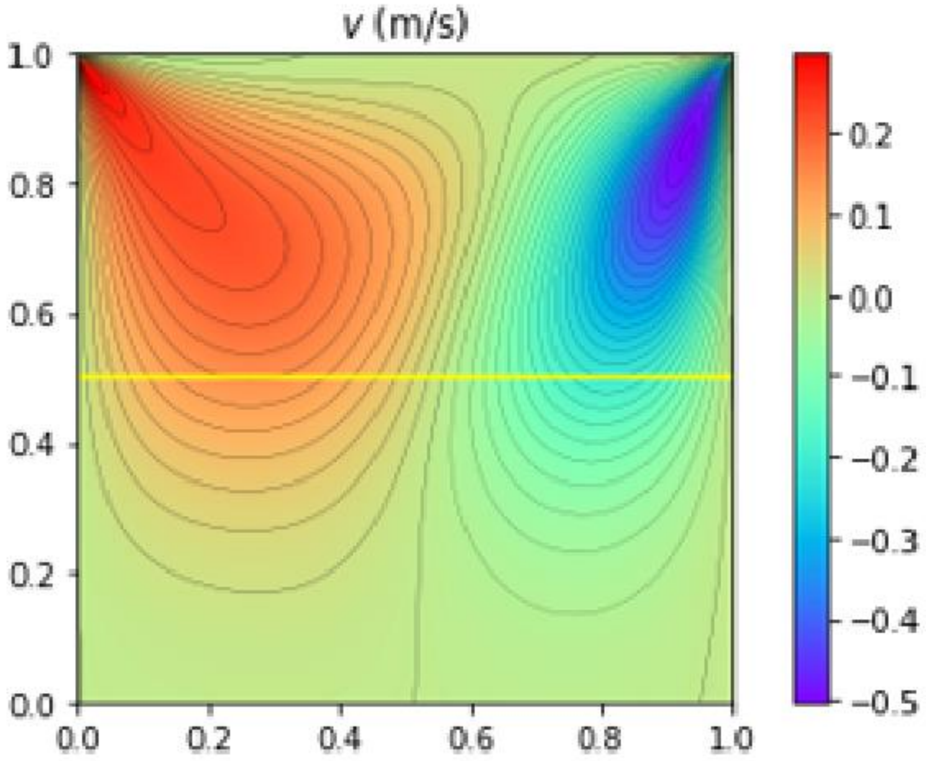


Figure 13. Lid Driven Cavity Flow for Re=100, Illustration of u & v velocity fields using 2d contour plots. The picture has been taken by the following research paper [1]

In Figure 13Figure 11 can be seen the solution for Re=100 in the form of a 2-dimensional contour plot as has been found in the following paper [1].

The formulation of Navier-Stokes equations that describe the problem is the following.

$$\begin{aligned}
&\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \\
&u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\varrho}\frac{\partial p}{\partial x} - \frac{\mu}{\varrho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0 \\
&u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\varrho}\frac{\partial p}{\partial y} - \frac{\mu}{\varrho}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0
\end{aligned} \qquad \textbf{Eq. 3}$$

where the dimensionless variables are,

x – coordinate along the x direction, 0 < x < 1

y – coordinate along the y direction, 0 < y < 1

u – velocity in x direction

v – velocity in y direction

p – pressure

μ – kinematic viscosity, [0.01, 0.0025, 0.001] m2/s for Re = [100, 400, 1000]

ρ – density, 1.0 kg/m

2.4 Static Piston Flow Problem

The Static Piston Flow problem is the most challenging problem compared to all the previous ones. We consider a square domain, shown in Figure 14, with all the four walls have a dimension of 1m. For the bottom, left and right walls the velocities u and v are always zero. Instead, for the top wall, the horizontal velocity u is everywhere 0 but the vertical velocity v at the injection position xInjection is 1 m/s for a short period of time when 0.1042<t<1.1156 ms. The phenomenon is transient but periodic and has a simulation period of 42.38 ms. The injection point in the top wall is a parameter which means that for different values of it, the solution of the following system of differential equations changes.

$$\begin{aligned}
&\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \\
&\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\varrho}\frac{\partial p}{\partial x} - \frac{\mu}{\varrho}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0 \\
&\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\varrho}\frac{\partial p}{\partial y} - \frac{\mu}{\varrho}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0
\end{aligned} \qquad \textbf{Eq. 4}$$

where the dimensionless variables are,

x – coordinate along the x direction, $0 < x < 1$

y – coordinate along the y direction, $0 < y < 1$

t – time, $0 < t < 42.38$ ms

u – velocity in x direction

v – velocity in y direction

p – pressure

μ – kinematic viscosity, 2.0e-6 m2/s

ρ – density, 1.0 kg/m

Given the above, it can be calculated the Reynolds number of the flow based on the injection as Re = u*L/v. The kinematic viscosity is v = 2e-6 [m2/s], so at near the injector (the top region) if we use L the injector hold width, which is L = (1/32) [m], we have injection velocity u = 1.0 [m/s], then we end up with Re = 1.0 * 1/32 / (2e-6) = 15625.

Goal of this problem is to try training a PINN that will solve and will learn all the solutions of the above system for different cases (injection points).

In Figure 14 and Figure 15 is illustrated the solution of the problem for Re = 15625 of u & v velocity field using Quiver and 2D Contour plots.

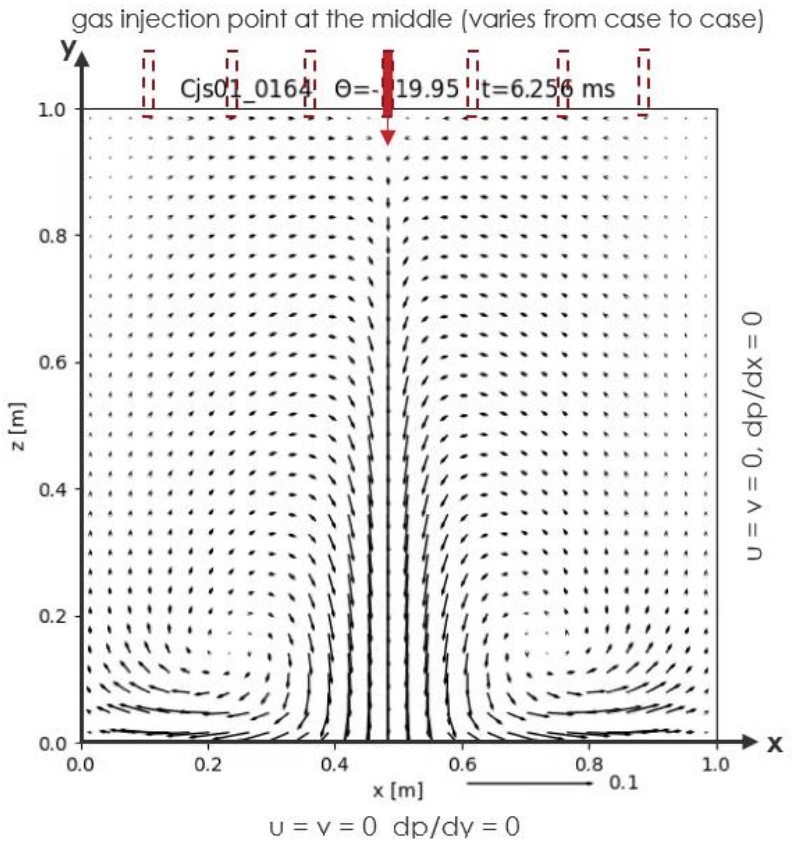
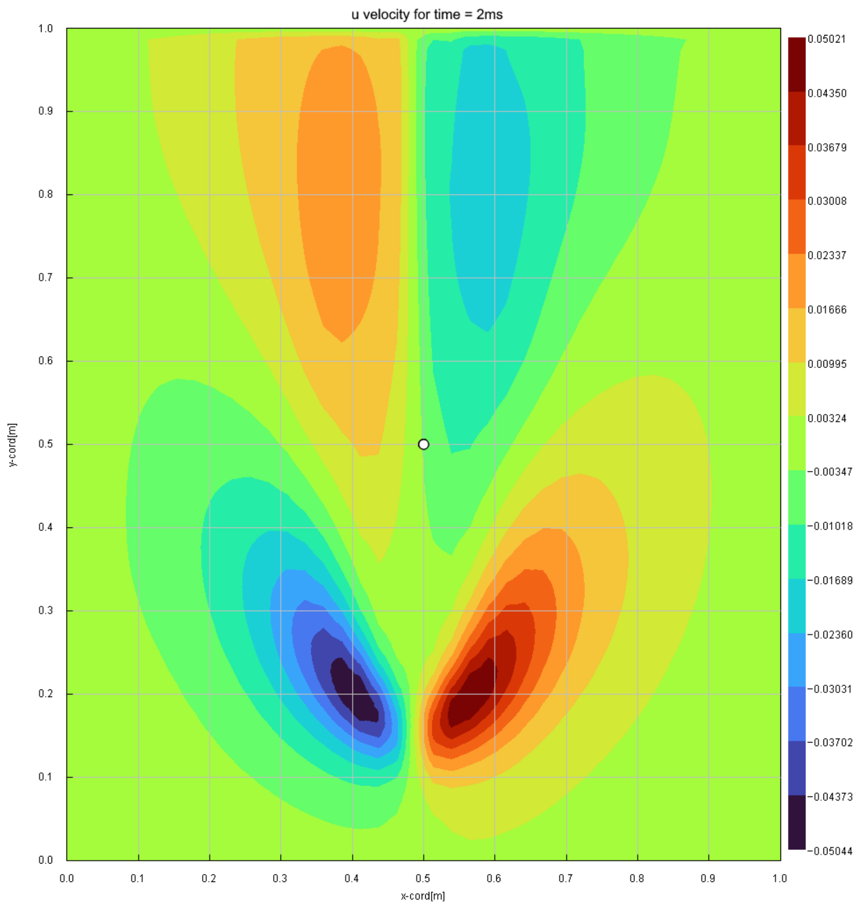


Figure 14. **Static Piston Flow for Re = 15625, Illustration of u & v velocity field using Quiver plot. This velocity field corresponds to time=6.256 ms and xInj=0.5m.**



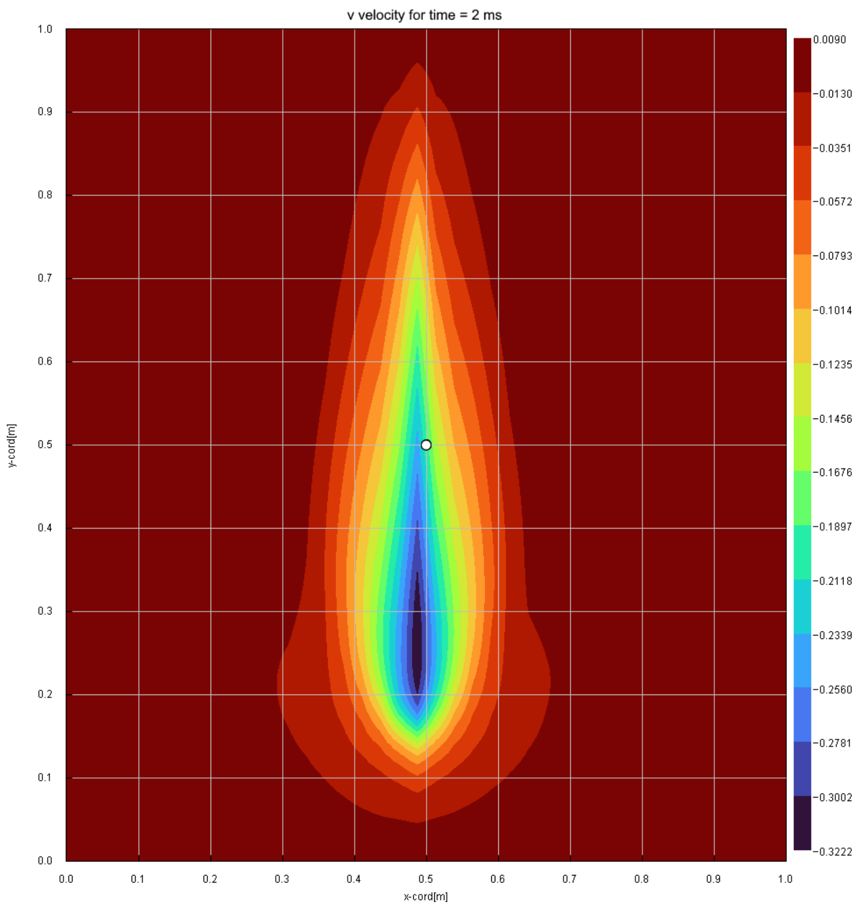


Figure 15. Static Piston Flow for Re = 15625, Illustration of u & v velocity field using 2d contour plots. This velocity field corresponds to time=2 ms and xInj=0.5m.

3 CHAPTER 3: Implementation and Detailed results

In this chapter we present the results that have been achieved for all the problems descripted in CHAPTER 2: Experiments. The Goal of all the problems was to approximate the solution of the physical system with a PINN that minimizes the physics error everywhere in the domain and at the same time satisfies with high accuracy the boundary and/or initial conditions. This practically means that all the PINNs have not used any known data for their training (unsupervised learning), instead in some cases that known data were available have been used only for validation purposes or for training MLP models using clearly supervised learning approaches which were used for comparison and visualization purposes. PINNs because of their characteristic to calculate high order derivatives of the outputs with respect to the inputs do not work at all when use activation functions such as ReLU or Leaky ReLU where the 2nd derivative is zero. For this reason and without loss of generality in all our experiments we use fully connected multi-layer PINNs that incorporate Tanh activation functions for the hidden layers and Linear activation functions for the output layer.

Modern Deep Learning applications, such as NLP and Computer Vision are dominated by the stochastic gradient descent 1st order optimizers, like SGD or Adam, mainly because of their ability to scale to millions of parameters and to avoid local minima assisted by their stochastic nature. Despite their advantages these algorithms many times suffer from slow convergence and insufficient accuracy in demanding physics based scientific and engineering scenarios. In contrary, 2nd order optimizers such as BFGS or Levenberg-Marquardt despite their superiority, in terms of accuracy and convergence speed, against 1st order optimizers, are practically limited only to medium size NNs (less than 10000 parameters). This is because of their increased computational complexity caused by the existence of the squared Hessian matrix with dimensions $\Theta x \Theta$ where Θ is the number of parameters in the NN.
However, the community has developed limited memory versions of 2nd order optimizers with the most popular algorithm to be the L-BFGS [10] followed by the newest addition of the AdaHessian [15].
In the field of PINNs, Issac Lagaris [2] reported that he achieved the best results using the BFGS among other algorithms. Raissi el al. [3] used for the first time very deep NNs of 10 hidden layers for solving the Burgers equation problem and for this purpose used as an optimization scheme the Adam optimizer as a warm starter followed by the L-BFGS for finetuning further the solution. The last years the above optimization scheme has become the dominant way for training PINNs. In this thesis, along with the previous scheme, we are going to follow a slightly different approach by using the BFGS exclusively or together with the Adam as a warm starter. These alternative approaches have been found to achieve superior results in accuracy with smaller NN architectures.

3.1 Results of Burgers equation

Burgers equation serves as an essential benchmark problem in PINNs. This equation was initially addressed by Raissi et al. [3] in their groundbreaking paper introducing PINNs. Although Burgers equation may seem straightforward, solving it is crucial for our research. By tackling this relatively simple problem, we can verify the accuracy of our implementation and validate our chosen methodology.

As has been explained in Eq. 1, the domain of the problem can be defined as follows:

$$\begin{aligned} & x \in [\mathbf{-1}, \mathbf{1}], t \in [\mathbf{0}, \mathbf{1}] \\ & \boldsymbol{u}(\mathbf{0}, \boldsymbol{x}) = -\boldsymbol{sin}(\boldsymbol{\pi x}), \\ & \boldsymbol{u}(\boldsymbol{t}, \mathbf{-1}) = \boldsymbol{u}(\boldsymbol{t}, \mathbf{1}) = \mathbf{0}, \end{aligned} \qquad \textbf{Eq. 5}$$

To cover sufficiently the domain must be sampled

- 50 points equally distributed in x axis within the range [-1, 1] for t=0 as initial conditions.
- 50 points equally distributed in t axis within the range [0,1] for x=-1 as boundary conditions.
- 50 points equally distributed in t axis within the range [0,1] for x= 1 as boundary conditions.
- 1000 collocations points uniformly distributed within the domain.

The PINN model that will approximate the solution takes as inputs the spatial variable x and time t and outputs the solution *u* as can be seen in **Error! Reference source not found.** . We utilized a PINN architecture consisting of 4 hidden layers, each containing 30 neurons with a hyperbolic tangent (tanh) activation function.

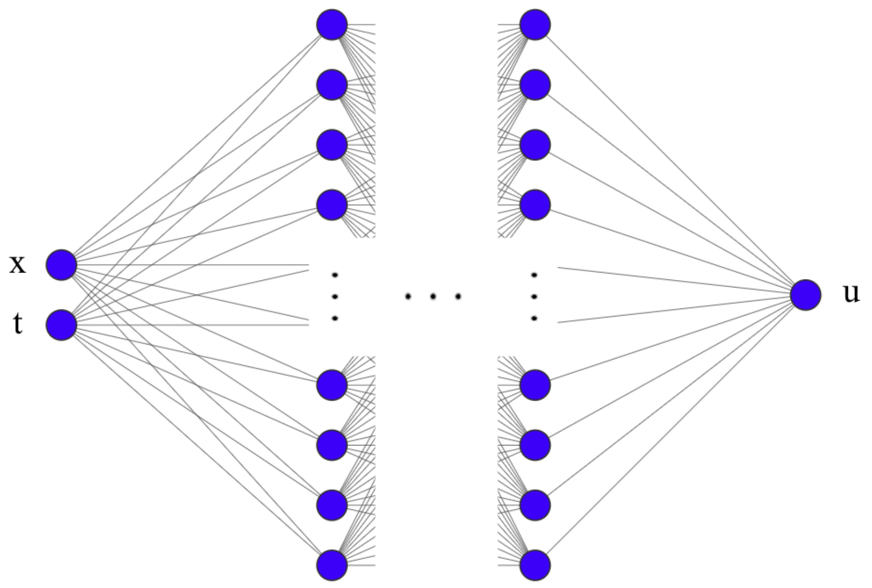


Figure 16. The PINN architecture that represents the solution of Burgers Equation.

Regarding the optimization, we followed the standard practice, and we used the Adam for 1000 iterations as a warm starter followed by the L-BFGS for about another 1000 iterations until converged. The training progress is reported in Figure 17 where can be seen the 2 different phases of the optimization and how the use of L-BFGS helps to minimize drastically the composed Mean Squared Error of Burgers equation. Finally, in Figure 18 is presented the predicted solution of Burgers equation using the trained PINN where can be clearly can be seen the formed shock wave at t=1.

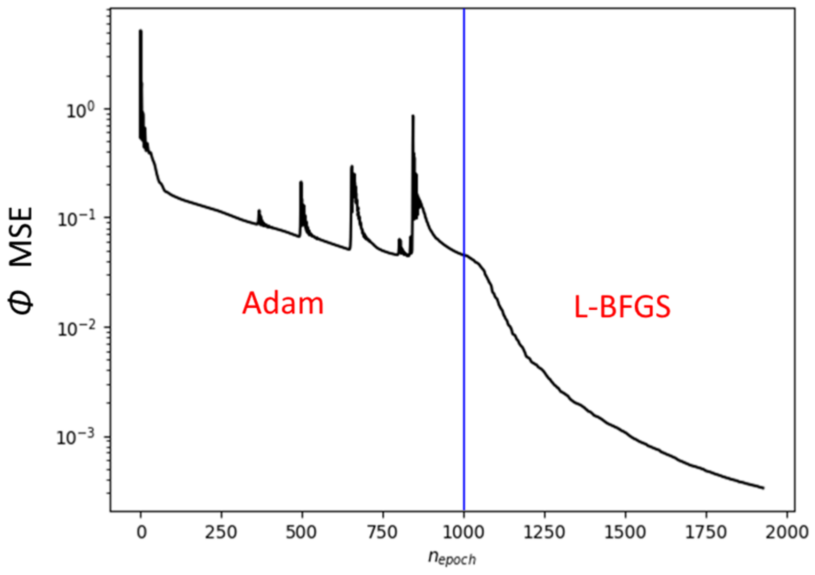


Figure 17. The composed Mean Squared Error of Burgers equation over the number of training epochs.

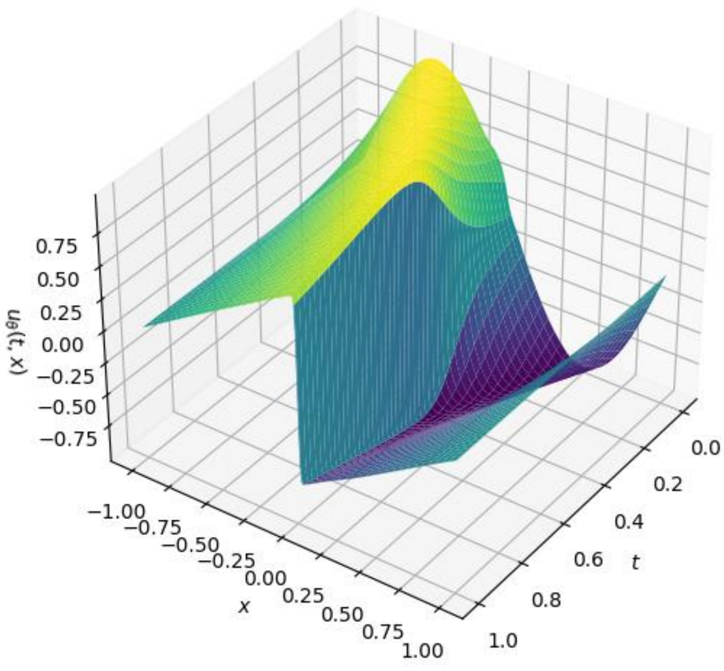


Figure 18. An illustration of the predicted solution of Burgers equation using PINN. Can be clearly seen the formed shock wave at t=1.

3.2 Results of Poisson equation

The second benchmark problem to be solved is Poisson equation. As mentioned in Poisson equation the formulation of the problem including the domain and the source terms were taken from the scientific paper “Physics-Informed Deep-Learning for Scientific Computing” by Stefano Markidis [13]. Our main purpose for solving the Poisson equation is to verify the results and so to build confidence in the overall framework and reduces the likelihood of errors in subsequent, more complex simulations.

As has been explained in Eq. 2 the domain of the problem can be defined as follows:

$$\begin{aligned} &x \in [\mathbf{0}, \mathbf{1}], \\ &y \in [\mathbf{0}, \mathbf{1}], \\ &u(\mathbf{0}, y) = \mathbf{0}, u(\mathbf{1}, y) = \mathbf{0}, u(x, \mathbf{0}) = \mathbf{0}, u(x, \mathbf{1}) = \mathbf{0} \end{aligned} \qquad \textbf{Eq. 6}$$

Our dataset consists of:

- 4x50 boundary points equally distributed in the boundaries of the square domain.
- 1000 collocations points uniformly distributed within the domain.

The PINN model that will approximate the solution takes as inputs the spatial variables x, y and outputs the solution u as can be seen in Figure 19**Error! Reference source not found.** . Similarly to the Burgers equation we utilized a PINN architecture consisting of 4 hidden layers, each containing 30 neurons with a hyperbolic tangent (tanh) activation function.

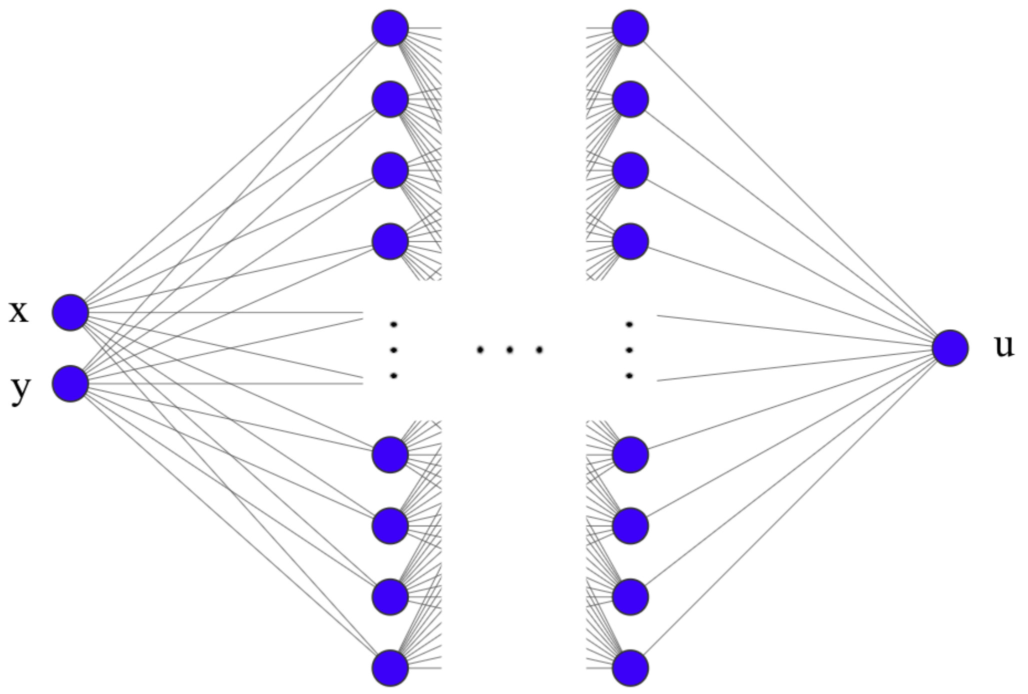


Figure 19. The PINN architecture that represents the solution of Poisson equation.

We trained the PINN using the Adam optimizer for 2,000 epochs, followed by BFGS optimizer for about 2000 iterations until converged. We used BFGS instead of L-BFGS because BFGS according to our experiments achieved much better accuracy than the limited memory version of it. In Figure 20 it is demonstrated the positive impact of BFGS to the optimization by reducing further the composed Mean Squared Error of Poisson equation by 3 orders of magnitude compared to Adam. While, in Figure 21 is presented the predicted solution of Poisson equation using the trained PINN drawn in a 3d contour plot. In the plot someone can see the multiple extrema caused by the various source terms introduced in the equation Eq. 2.

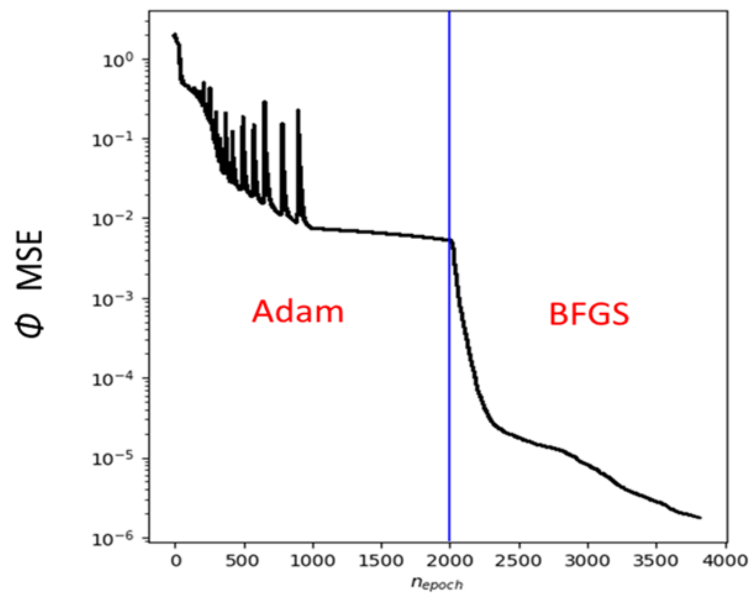


Figure 20. The composed Mean Squared Error of Poisson equation over the number of training epochs.

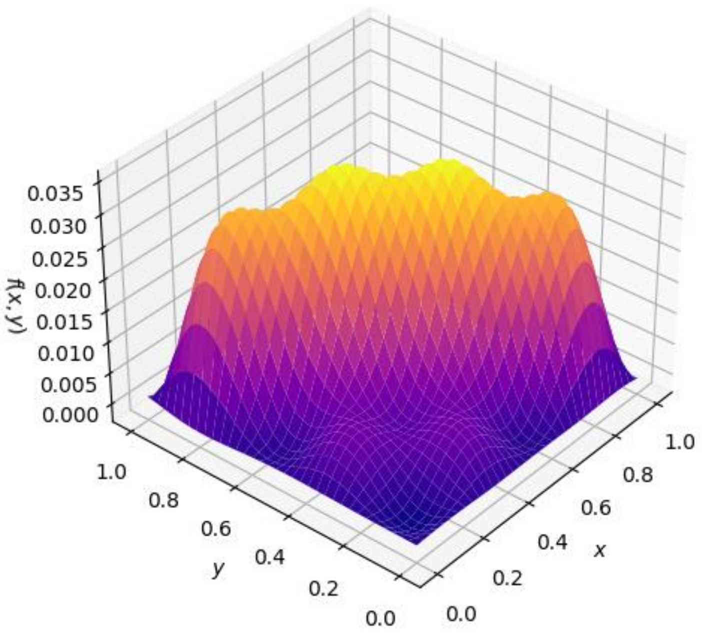


Figure 21. An illustration of the predicted solution of the Poisson equation using PINN.

3.3 Results of Lid-Driven Cavity Flow problem

The lid-driven cavity flow problem is a fundamental benchmark in computational fluid dynamics, characterized by its complex velocity field and recirculating flows. This study aims to investigate the application of PINNs to solve this challenging fluid mechanics problem across various Reynolds numbers (Re), which pose significant challenges due to increased turbulence and complexity in the flow field.

Our research focused on three primary objectives:

1. Comparing our PINN solution for Re = 100 with the best results of the "On Physics-Informed Deep Learning for Solving Navier-Stokes Equations" [1] research paper.
2. Assessing the applicability of PINNs to higher Reynolds numbers, specifically Re=400 and Re=1000, and comparing the results with established numerical methods, as presented in the seminal work by Ghia et al. [16].
3. Investigating the impact of neural network architecture, boundary condition density, and collocation grid size on the accuracy and efficiency of PINN solutions across these Reynolds regimes.

3.3.1 Re = 100

The research paper reported the best results using a PINN architecture with 10 hidden layers, each containing 40 neurons. The model utilized 2500 boundary condition points and a 200x200 grid of collocation points, resulting in 40,000 total points. Training was conducted using Adam optimization for 5000 iterations, followed by L-BFGS for an additional 2451 iterations until convergence was achieved.

To demonstrate the significance of optimizer training capabilities, we employed BFGS optimization exclusively until convergence after 2420 iterations. We utilized a smaller PINN architecture consisting of 5 hidden layers, each containing 40 neurons, to minimize memory consumption and computational overhead associated with Hessian matrix calculations. By reducing the number of parameters in the PINN, we opted to evaluate the loss function using fewer boundary conditions and collocation points. Specifically, we chose 4x50 boundary condition points and a 50x50 grid of collocation points. The PINN model takes as inputs the spatial variables x and y and outputs u and v velocities, followed by pressure p as can be seen in Figure 22

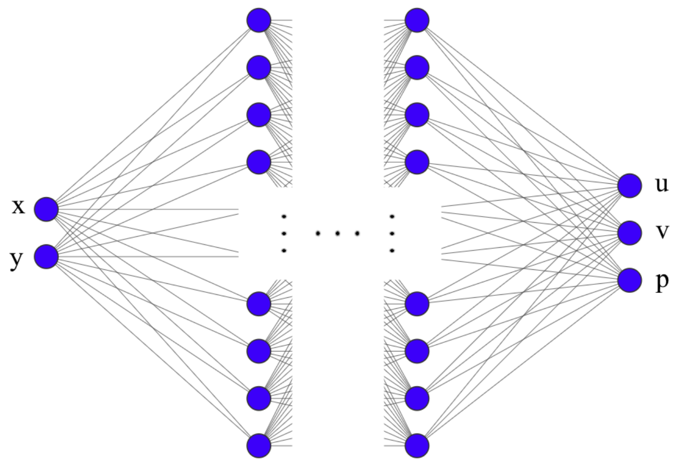


Figure 22. The PINN architecture that represents the solution of Lid Driven Cavity Flow problem.

In Figure 23. is reported the Mean Squared Error for both physics and Boundary conditions loss terms over the training epochs.

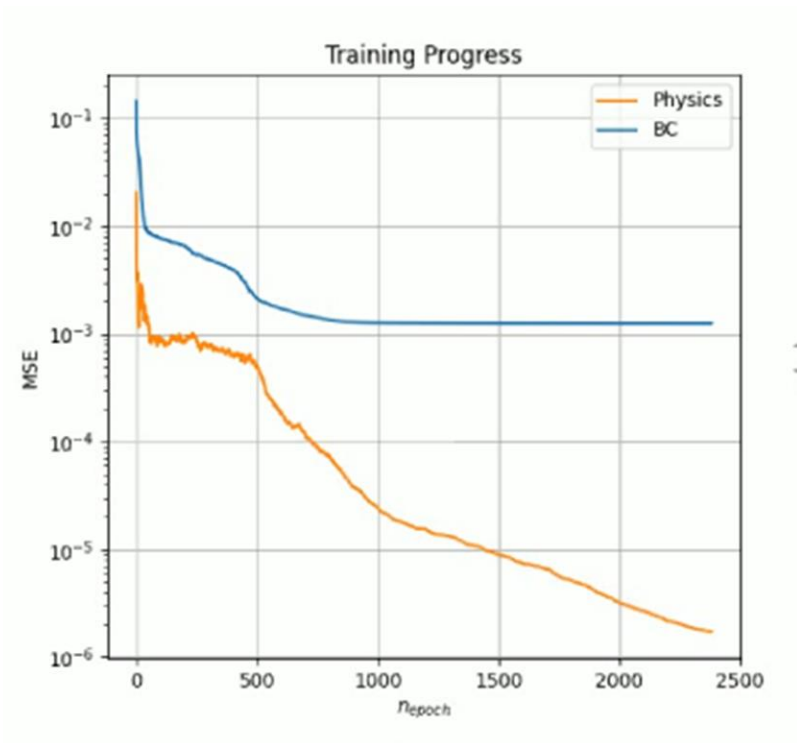


Figure 23. The Physics & Boundary Condition Mean Squared Errors separately reported for Re=100 over the number of epochs.

Using the trained PINN, a set of visualizations has been generated to evaluate the calculated solution (Figure 24). On the left-hand side, the two 2-dimensional contour plots represent the predicted solution of the u & v velocity field across the entire domain. The yellow vertical and horizontal lines indicate the midpoint sections of these solutions. On right-hand side can be found the predicted solution calculated for x=0.5m and y=0.5m respectively. The red dots represent the exact solution for Re=100 which were originally presented in the study by Ghia et al [16].

Accordingly, in Figure 25 a set of comparison plots is presented that showcase the superior accuracy achieved by our PINN, despite its relatively shallow architecture compared to the deeper networks suggested in the reference paper. This outcome strongly suggests that many researchers may employ unnecessarily deep architectures instead of focusing on optimizing their models with better algorithms. Our results indicate that achieving high performance doesn't necessarily require extremely deep neural networks, but rather effective optimization techniques. This finding challenges the conventional wisdom that deeper networks always lead to better results, highlighting the potential benefits of exploring alternative optimization strategies in PINN implementations.

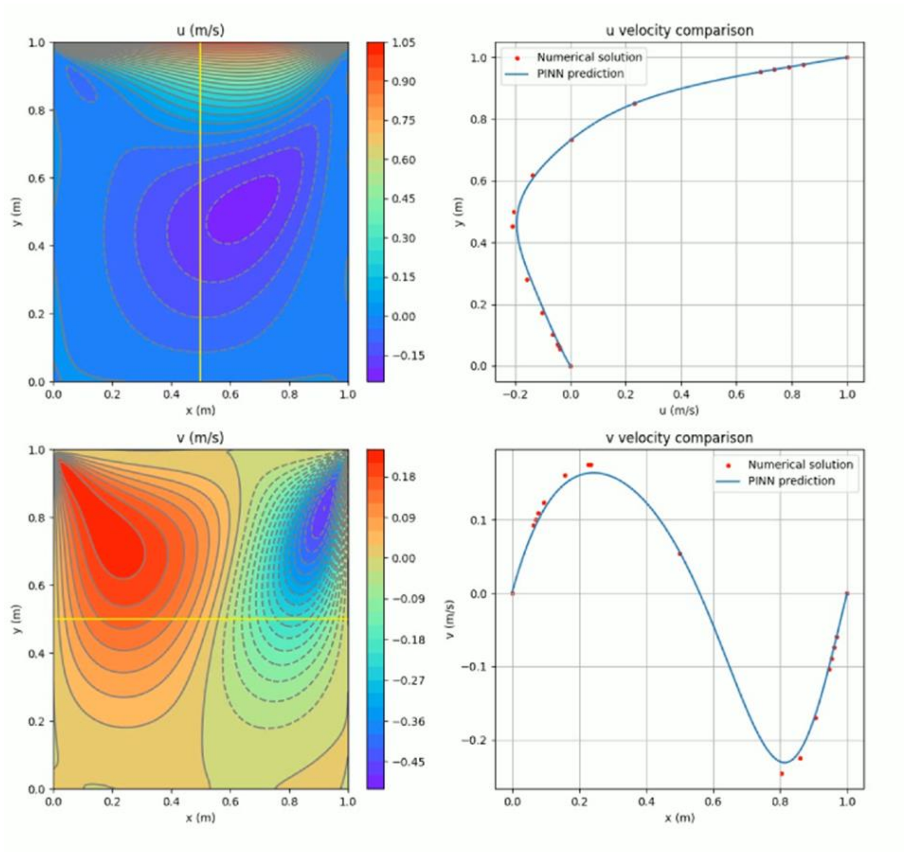


Figure 24. **The predicted by the PINN u & v velocity field for Re=100 illustrated as 2d contour plots on the left plot column. On the right plot column can be seen the PINN prediction for the mid-section values compared with the numerical solution (red dots)**

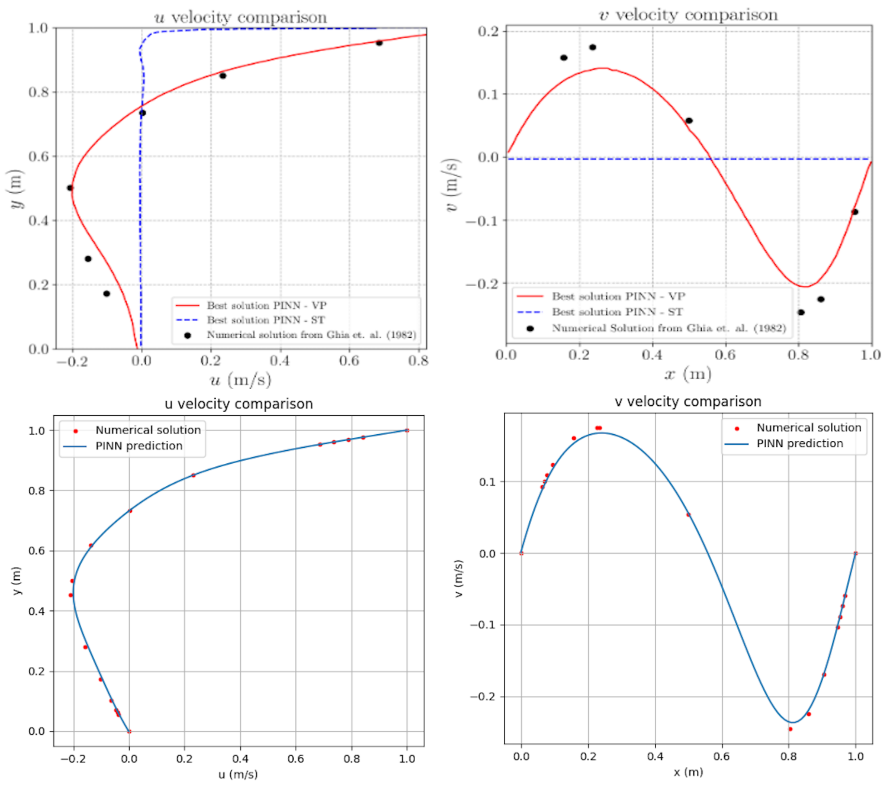


Figure 25. **On the top plot row can be seen the best results achieved for Re=100 by the research paper while on the bottom plot row can be seen the results using our trained PINN.**

3.3.2 Re = 400

Trying to assess further the applicability of our PINN methodology has been decided to try solving the problem for a higher Reynold number equal to 400. The solution of the Lid Driven Cavity Flow for Re = 400 is known to be more difficult because of the high nonlinearity that is caused by the increased turbulence.

Similarly to our previous experiment for Re = 100, a PINN architecture it has been utilized consisting of 5 hidden layers, each containing 40 neurons, a 4x50 boundary condition points and a 50x50 grid of collocation points. Several experiments have been conducted without being possible to get an accurate solution. After many trials and error, the problem has been solved successfully by increasing the boundary condition points to 4x100 and the grid of collocation points to 128x128.

Figure 26 illustrates both the Physics and Boundary conditions loss terms where can be seen between iterations 100 and 300 the competing nature of PINN optimization problem where when the BC error decreases but the Physics error increases.

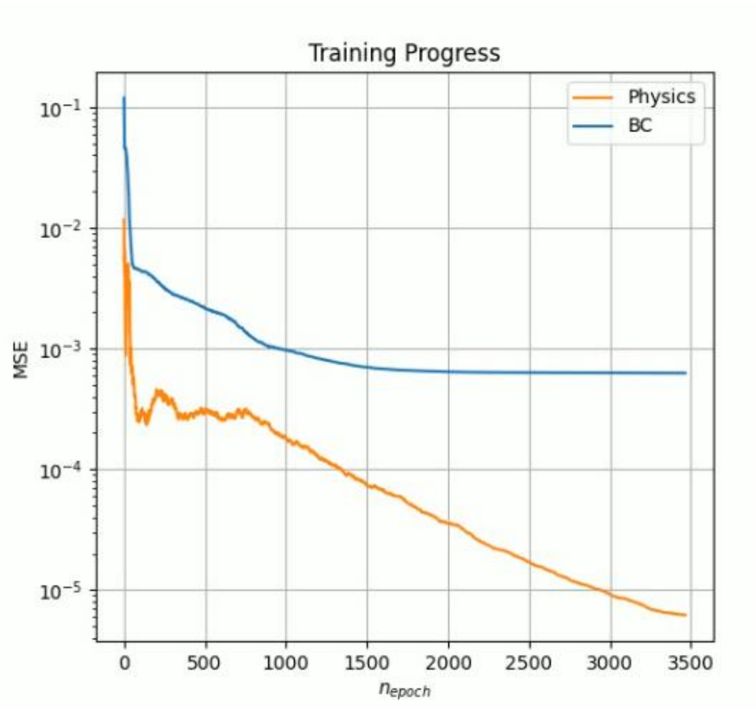


Figure 26. The Physics & Boundary Condition Mean Squared Errors separately reported for Re=400 over the number of epochs.

Despite the success of our PINN methodology for R=400, in Figure 27 it is obvious that PINN has difficulty to capture with high accuracy the dynamics.

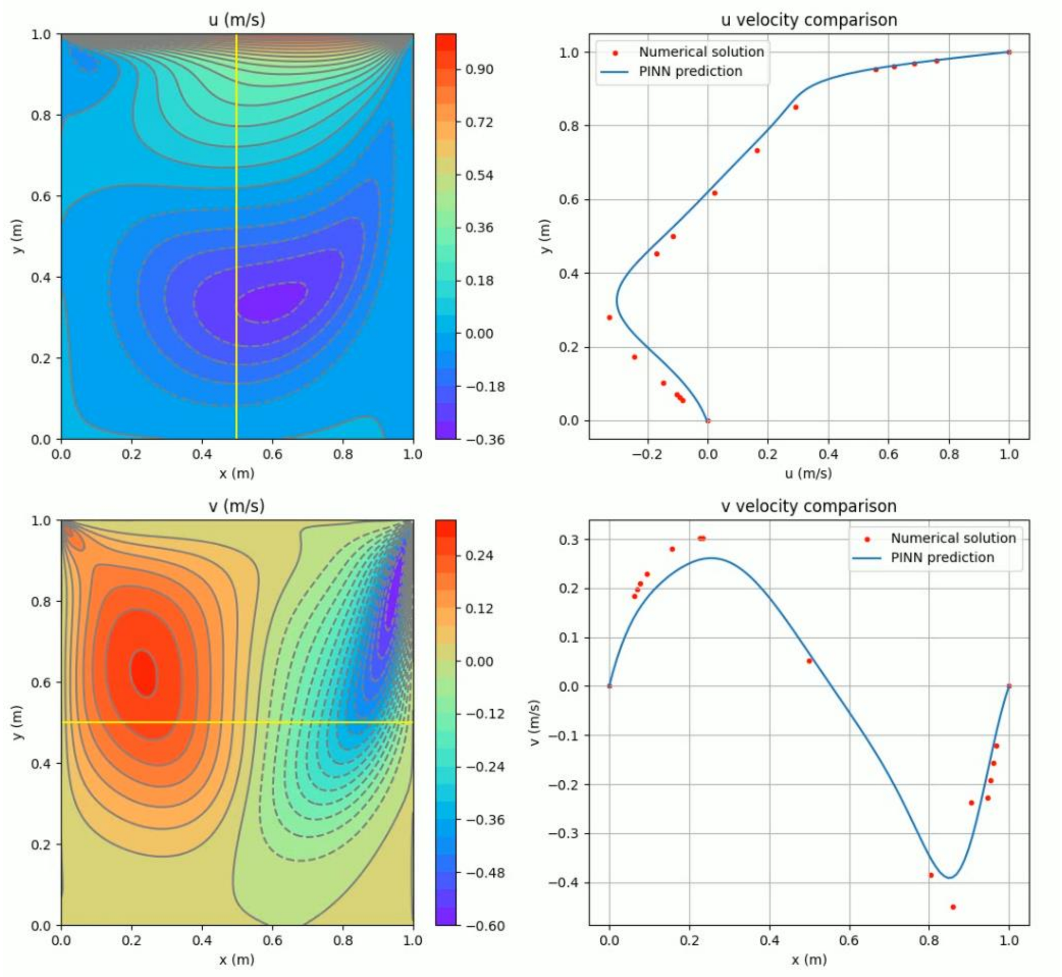


Figure 27. The predicted by the PINN u & v velocity field for Re=400 illustrated as 2d contour plots on the left plot column. On the right plot column can be seen the PINN prediction for the mid-section values compared with the numerical solution (red dots).

3.3.3 Re = 1000

As a final more challenging step we decided to evaluate PINNs methodology in solving the problem at a Reynolds number (Re) of 1000. For solving the Navier-Stokes equations for Re=1000, has been used initially a PINN with the same architecture, boundary condition points and collocation points as for the lower Re=400 case. Despite numerous attempts, we were unable to obtain satisfactory results using this approach. In an effort to overcome these challenges, we explored several strategies such as, employing larger neural network architectures, increasing the number of boundary condition points and implementing a denser grid of collocation points. However, regardless of these adjustments, we were consistently unable to achieve a successful solution to the problem. It seems that, the complexity of the problem at higher Reynolds numbers appears to be beyond the capabilities of our current implementation, necessitating further research or alternative approaches to tackle this challenging scenario.

Figure 28Figure 26 shows how both the Physics loss and Boundary conditions loss terms for Re=1000 evolve over time (number of epochs). Finally, in Figure 29Figure 27 we see an illustration of the failure of PINNs to approximate the numerical solution provided by Ghia et al [16].

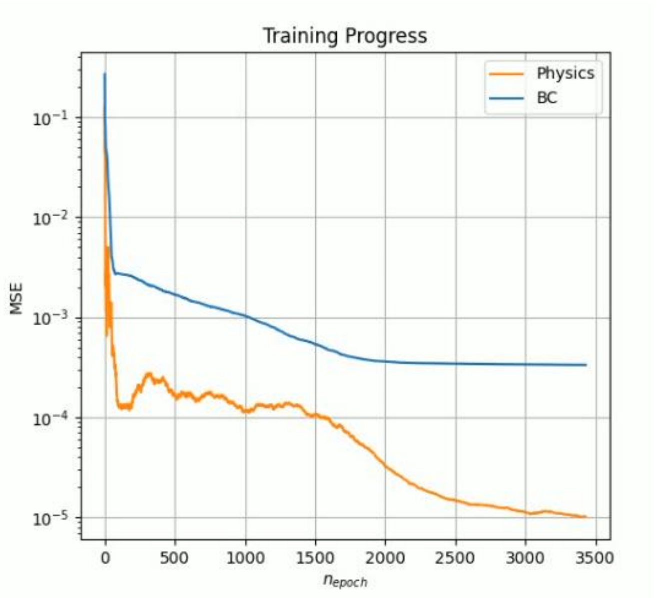


Figure 28. **The Physics & Boundary Condition Mean Squared Errors separately reported for Re=1000 over the number of epochs.**

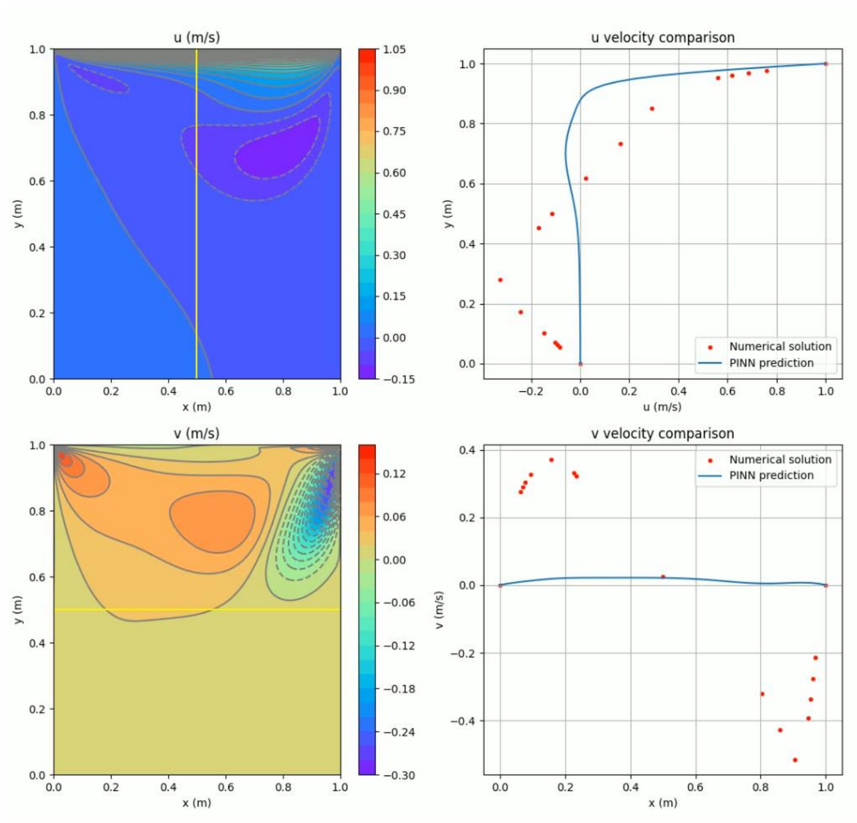


Figure 29. **The predicted by the PINN u & v velocity field for Re=1000 illustrated as 2d contour plots on the left plot column. On the right plot column can be seen the PINN prediction for the mid-section values compared with the numerical solution (red dots).**

3.4 Results of Static Piston Flow problem

The Static Piston Flow problem serves as our final and most demanding evaluation of the PINNs within this thesis. In Static Piston Flow Problem, we determined that the Reynolds number of the flow was 15625, indicating a chaotic turbulent flow regime. This problem involves a four-dimensional domain, including x and y spatial coordinates, time t, and the position of injection, representing various initial conditions applied to the top wall of the piston. Static Piston Flow is not a standard benchmark problem found in literature, making it difficult to obtain solutions from public sources. Typically, such complex problems are solved using computational fluid dynamics (CFD) software like OpenFOAM or similar tools. In our case, the numerical solution was provided by Gamma Technologies, a leading company in automotive computer-aided engineering (CAE), utilizing the Stable Fluid method.

The problem was solved seven times, each corresponding to a different injection position as shown in Figure 14. Our dataset consists of a four-dimensional regular grid with the following dimensions:

- [X * Y * T * #Of Injection Pts] = [32 * 32 * 250 * 7]
    - Total collocation points: 1,792,000
    - Boundary condition points: 224,000
    - Initial condition points: 7,168

For each point in the dataset, we know the values of the system states U, V, and P.

We intend to apply PINNs (Figure 30. The PINN architecture that represents the solution of Static Piston Flow problem.) in an unsupervised learning manner. Therefore, the provided dataset will not be used to train the neural network parameters. Instead, it will serve to evaluate the validation data error, providing a metric to assess whether the PINN can accurately approximate the problem's solution.

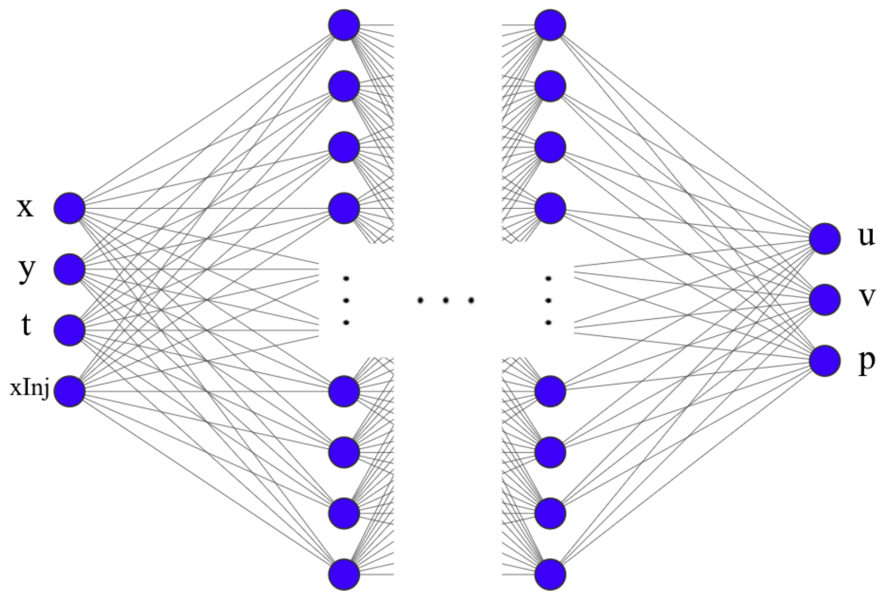


Figure 30. The PINN architecture that represents the solution of Static Piston Flow problem.

Similarly, as our previous experiments in Lid Driven Cavity Flow, we initially relied solely on the BFGS optimization method until convergence. Our approach utilized a PINN architecture featuring five hidden layers, each containing 40 neurons. Despite achieving low levels of physics and initial/boundary condition errors, the PINN consistently struggled to significantly reduce the data validation error. We conducted further experiments, exploring various combinations of neural network architectures and optimization techniques, including incorporating the Adam optimizer as a warm starter. Unfortunately, these additional training attempts yielded similar results, maintaining the error pattern illustrated in Figure 31 throughout all iterations.

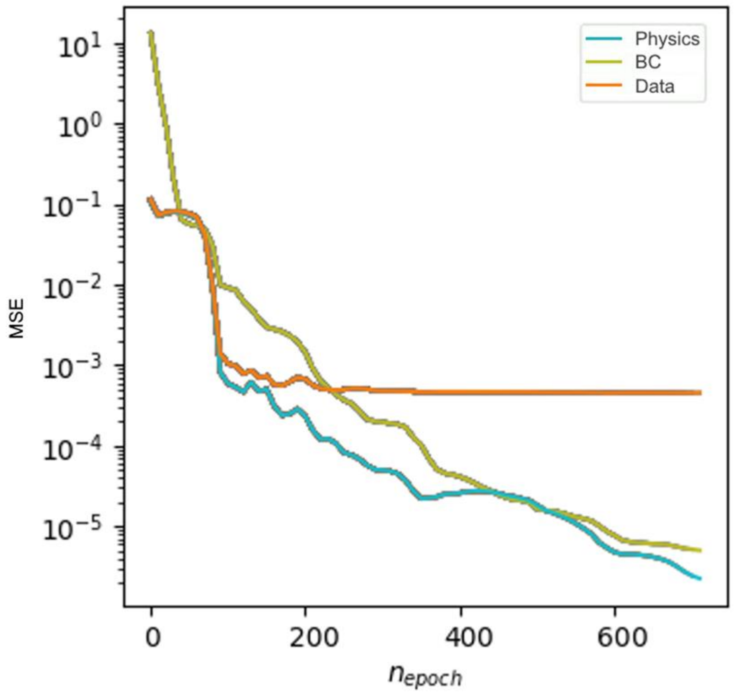


Figure 31. The Physics, Boundary Condition and Data validation Mean Squared Errors separately reported over the number of epochs. Illustrates the weakness of the PINN to reduce the data validation error while the Physics & Boundary Condition error have achieved low values.

Given the complexity of the problem at hand, some might argue that the current neural network architecture of 5 hidden layers lacks sufficient capacity to fully capture the underlying dynamics of the system. To investigate this hypothesis, we've chosen to leverage the available dataset and employ a traditional supervised learning approach with the existing neural network architecture. This strategy allows us to assess whether the network's performance improves when trained using a more conventional method, potentially revealing insights about weaknesses and limitations of PINN methodology.

We conducted the following training scenarios using the corresponding pairs of NN architectures and optimizers.

- 4 Hidden layers, 30 neurons each using Adam
- 4 Hidden layers, 30 neurons each using BFGS
- 4 Hidden layers, 20 neurons each using Levenberg-Marquardt

The Machine Learning Assistant (MLA) tool from Gamma Technologies was utilized for this study. All training sessions were conducted for 1000 epochs to ensure a fair comparison. Figure 32. The Root Mean Squared Errors achieved by Adam, BFGS and LM using supervised learning against the PINN.presents the Root Mean Squared Errors (RMSE) for the vertical velocity across various training methods. Upon analyzing the results, we observe that supervised learning techniques consistently outperform the PINN method by at least one order of magnitude in terms of accuracy. Notably, both BFGS and Adam were employed to optimize the same neural network architecture, yet BFGS demonstrates superior performance compared to Adam, indicating potential premature stagnation in Adam's optimization process. LM stands out as the clear winner among the tested optimizers, achieving accuracy one order of magnitude higher than its counterparts (Figure 33. The prediction of v velocity by an MLP (4x20) trained by the LM using supervised learning.and Figure 34) and two orders of magnitude better than PINN. Furthermore, LM accomplishes this level of accuracy using a smaller architecture and exhibits faster convergence rates compared to the other two optimizers.

The significant advantage of the Supervised Learning approach over PINN suggests that the primary cause of PINN's underperformance is not related to the neural network's ability to learn, but rather stems from the inherent challenges in optimizing the more complex optimization landscape of PINN training process. PINN requires satisfying both data (boundary/initial conditions) fitting and physical constraint satisfaction simultaneously. This dual objective often leads to more challenging optimization problems compared to standard supervised learning tasks.

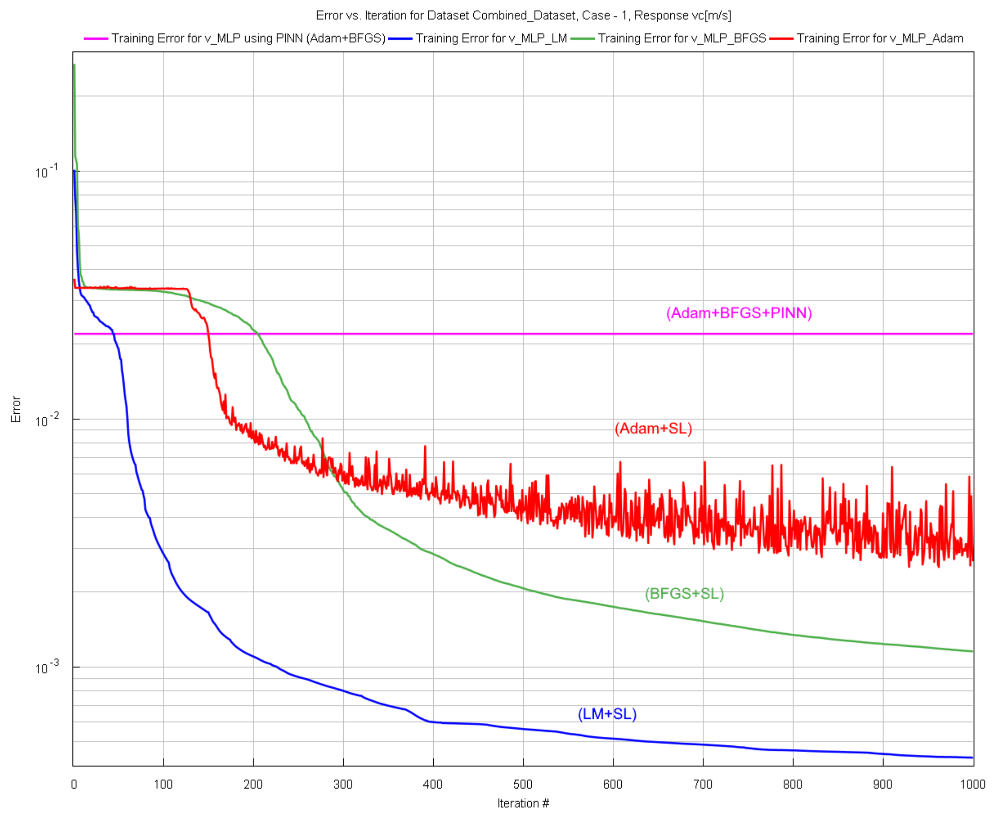


Figure 32. The Root Mean Squared Errors achieved by Adam, BFGS and LM using supervised learning against the PINN.

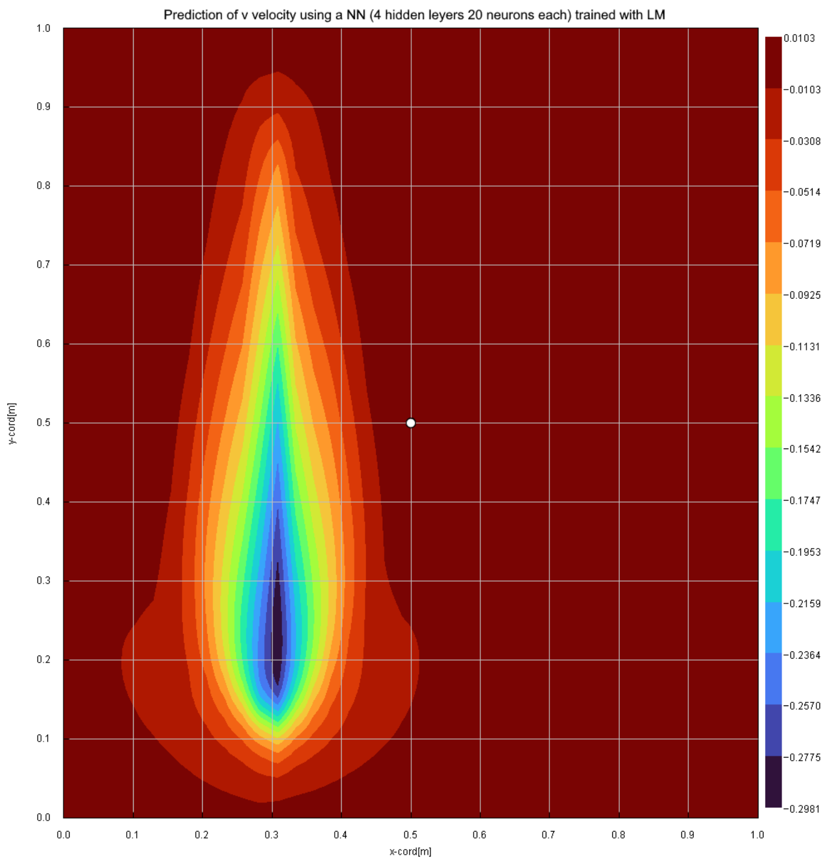


Figure 33. **The prediction of v velocity by an MLP (4x20) trained by the LM using supervised learning.**

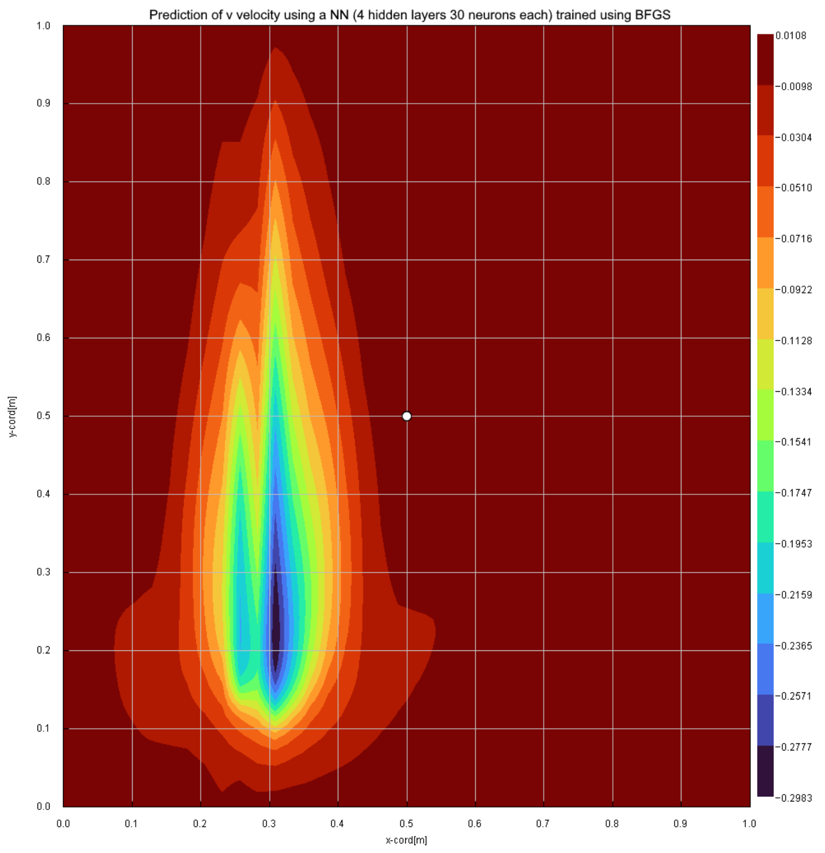


Figure 34. **The prediction of v velocity by an MLP (4x30) trained by the BFGS using supervised learning.**

4 Chapter 4: PINN’s pathologies – Possible Enhancements

PINNs have emerged as a promising tool for PDEs and modeling complex physical systems. Despite their potential, PINNs can fail in various scenarios. Training a PINN is a hard optimization problem that under some conditions can lead to a Neural Network that does not represent a physical solution. This chapter indicates the pathologies and explores the primary reasons behind PINN failures, providing insight into the challenges faced by this innovative approach.

4.1 PINNs’ issues

4.1.1 Automatic differentiation is not without a cost

In PINNs, Automatic differentiation (AD) is used to compute derivatives of the neural network output with respect to the inputs. This is essential for enforcing physical laws and boundary conditions within the network. AD allows for efficient computation of gradients required for optimization algorithms. It eliminates the need for manual derivation of analytical derivatives, which would be impractical for complex differential equations and neural networks.

However, the incorporation of AD into PINNs leads to an increased computational effort compared to a more classic Supervised Learning scenario:

Each calculation of a 1st order derivative with respect to an input doubles the size of the generated computational graph which affects accordingly the computational effort and the memory consumption as well. Calculating a 2nd order derivative with respect to an input demands four times more calculations and memory than a simple forward propagation of a classic MLP in Supervised Learning. Considering that the loss function often involves multiple evaluations of the network and its derivatives, someone can easily understand that the computational cost and memory consumption become a very considerable factor in PINN’s training.

4.1.2 Overfitting and Under-constrained Optimization

A fundamental issue in PINN implementation is the risk of overfitting and under-constrained optimization. When the neural network's capacity exceeds the information provided by the collocation and boundary condition points, it may memorize noise rather than learn meaningful patterns. This phenomenon is particularly pronounced in sparse sampling regimes, where insufficient constraints allow the network to converge to unphysical solutions despite achieving low training loss values. On the other hand, this problem could be mitigated by increasing the sampling density, but this would make PINNs suffer more from the curse of dimensionality losing so one advantage against the traditional numerical methods.

4.1.3 Competing optimization terms

The main reason PINNs suffer from competing optimization terms is due to the nature of their objective function. Typically, a PINN's loss function consists of two primary components:

- Boundary Conditions loss term: Encourages the network to fit the given boundary conditions points accurately.
- Physics loss term: Ensures that the predicted solutions satisfy the underlying physical laws.

These two terms can vary significantly in scale or in error landscape and often compete during the training process. The competition leads to a hard multimodal optimization problem which is difficult to be solved.

Minimizing the boundary conditions loss term may lead to solutions that don't perfectly adhere to the physical laws. Conversely, strictly enforcing the physics loss term might result in poor fitting of the actual boundary conditions. If during PINN optimization one of these terms is not minimized sufficiently the solution found will be sub-optimal or even unphysical.

4.1.4 Sensitivity to Hyperparameters

PINNs exhibit high sensitivity to various hyperparameters, including network architecture, optimization algorithms, and collocation point selection. Finding the optimal configuration for these parameters can be challenging, especially for complex problems. Suboptimal choices may lead to poor performance or failure to converge. Additionally, even for an optimal configuration found the solution can vary drastically for different training trials. All the above highlight the need for systematic approaches to hyperparameter tuning.

4.1.5 Difficulty in Capturing Complex Dynamics with large domains

PINNs face significant obstacles when applied to certain classes of physical problems characterized by intricate dynamics. These challenges arise primarily in systems that exhibit pronounced nonlinearity, broad spectral energy distributions, and heightened sensitivity to initial conditions [17] [18].

Two paradigmatic examples of such complex systems are:

1. Kuramoto-Sivashinsky Equation: This partial differential equation models various physical phenomena, including flame propagation and fluid flow. It is renowned for its chaotic behavior, presenting a formidable challenge for PINNs due to its highly nonlinear dynamics and sensitivity to initial conditions.

2. Navier-Stokes Equations in Turbulent Regime: These equations govern fluid motion and heat transfer. When applied to turbulent flows, they exhibit complex, multiscale phenomena characterized by broadband energy spectra. The chaotic nature of turbulence poses significant difficulties for PINNs in capturing both large-scale structures and small-scale fluctuations accurately.

4.2 PINNs' enhancements

4.2.1 Coupled Automatic and Numerical differentiation

Coupled-Automatic-Numerical Differentiation Method as per Pao-Hsiung Chiu et al [19] has been presented as one of the candidate solutions for handling the Overfitting and Under-constrained Optimization problem of PINNs.

To better understand the nature of this problem one can think the following scenario of the plot in Figure 35. Is given a solution of a hypothetical PDE which is represented by the black curve. A training process of a PINN that will evaluate the error in the following boundary and collocation points (dotted) using AD is possible to lead to the paradox of an unphysical solution (magenta) that satisfies almost to machine precision the PDE and the boundary conditions.

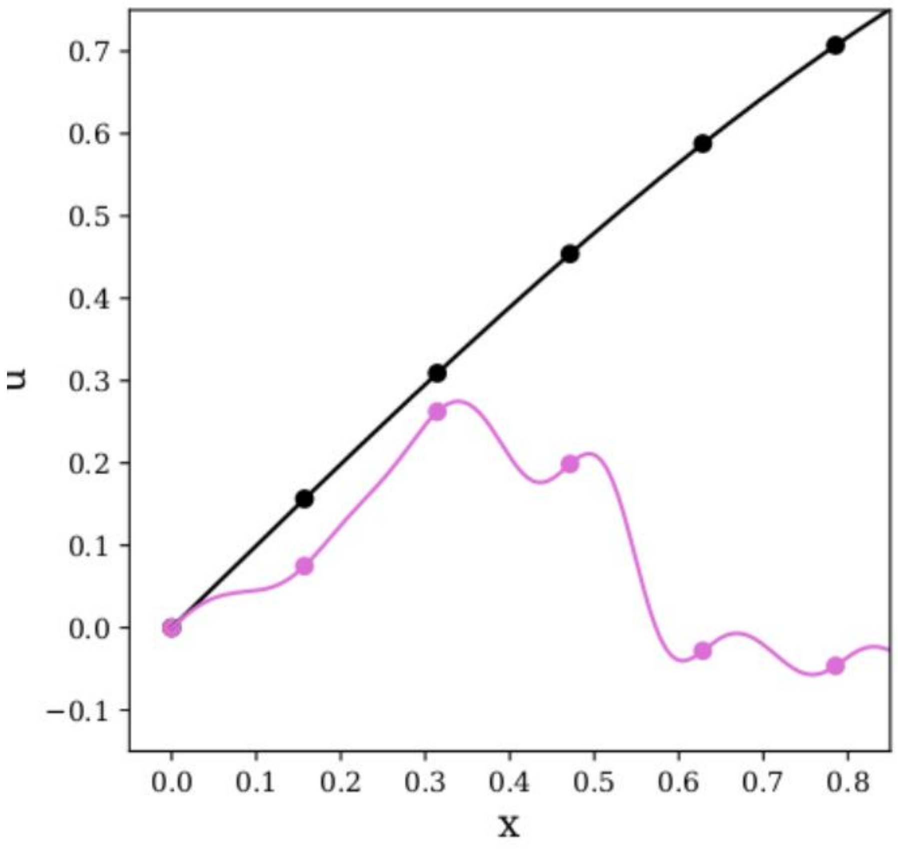


Figure 35. The paradox of an unphysical solution (magenta) that satisfies almost to machine precision the PDE and the boundary conditions [19].

The authors to alleviate this issue, they employed Numerical Differentiation to replace AD for the computation of differential operators required in PINNs.

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \approx \frac{\hat{u}(x + \mathbf{\Delta}x; w) - \hat{u}(x - \mathbf{\Delta}x; w)}{\mathbf{2\Delta}x}$$

$$\frac{\partial^2 \boldsymbol{u}}{\partial \boldsymbol{x}^2} \approx \frac{\hat{u}(x + \mathbf{\Delta}x; w) - \mathbf{2}\hat{u}(x; w) + \hat{u}(x - \mathbf{\Delta}x; w)}{\mathbf{\Delta}x^2} \quad \textbf{Eq. 7}$$

The numerical differentiation method computes differential operators using nearby support points around each collocation point. This novel approach, termed n-PINNs, seeks to adjust gradient behaviors during training within localized regions of the solution space, rather than focusing solely on individual collocation points. Applying n-PINNs to various benchmark problems resulted in enhanced accuracy. These experiments demonstrated that n-PINNs can consistently produce reliable solutions across both sparse and dense sampling scenarios.

However, Numerical differentiation despite the positive impact to PINNs training, it is known that introduces error in the calculation of the derivative terms of the PDE. The authors going even further tried a Coupled Automatic and Numerical differentiation method for calculating the derivatives that gave even better accuracy levels in PINNs training as can be shown in Figure 36.

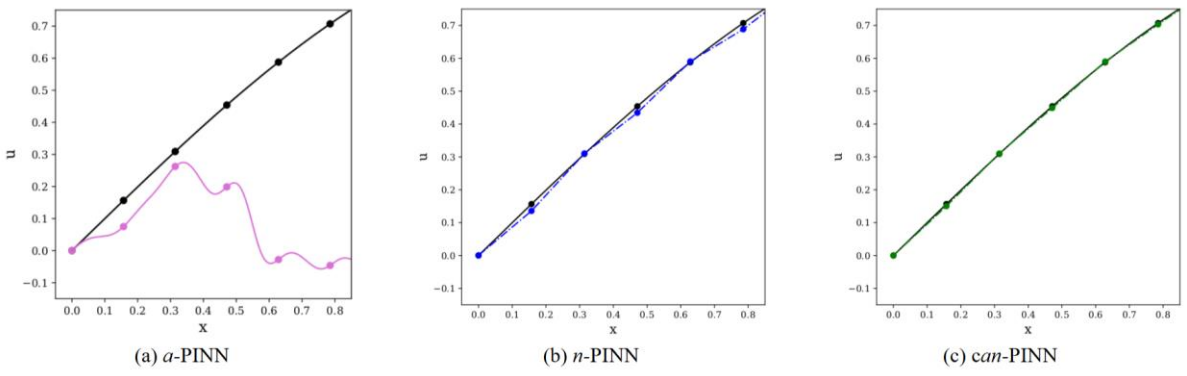


Figure 36. **A qualitative comparison plot of the 3 different differentiation approaches [19].**

4.2.2 Respect Temporal & Spatial Causality

Sifan Wang et al [20] in "Respecting causality is all you need for training physics-informed neural networks" claim that classic PINNs during their training do not respect spatio-temporal causal structure which is embedded to the evolution of dynamical systems. The authors believe that this fundamental limitation lead PINN models to erroneous solutions.

To grasp this concept, consider how traditional numerical techniques approach solving Partial Differential Equations (PDEs). These methods generally employ sequential algorithms that discretize time in a specific manner. The key point here is that the solution at any given time (t) must be fully determined before attempting to approximate the solution at the next time step (t + Δt). In PINNs instead, the PDE is treated as a global optimization problem that should be solved at once to the whole time or spatial domain. Such an optimization problem that does uses incorrect local state information is very difficult to be solved or even to converge to a valid physical solution.

The research paper tackles this issue by introducing a straightforward modification to the loss function used in PINNs as follows:

$$\boldsymbol{L_r(\theta) = \frac{1}{N}\sum_{1}^{N_t} w_i L_r(t_i, \theta)} \quad \textbf{Eq. 8}$$

Where w_i is

$$\boldsymbol{w_i = exp\left(-\varepsilon \sum_{k=1}^{N_t} L_r(t_k, \theta)\right)} \qquad \textbf{Eq. 9}$$

From these equations, we can deduce that the weight (wi) assigned to the current time step's residual loss is inversely related to the magnitude of the cumulative residual losses accumulated from previous time steps. As a result, the current time step's loss function $L_r(t_i, \theta)$ will not be minimized unless all previous residuals Lr (t, θ) decrease to some small value such that w_i becomes large enough.

4.2.3 Levenberg Marquardt for PINNs

In the following paper [21] "Shallow Physics Informed Neural Networks Using Levenberg-Marquardt Optimization" the researchers are trying to examine the potential use of Levenberg-Marquardt (LM) as a candidate game changer for the training of PINNs. So far, the most research on PINNs is focused on finding novel NN architectures or modifying the loss function accordingly to achieve better results in accuracy and training robustness. Based on their research it is established that the choice of the training algorithm is as important as the architecture and loss function. Traditionally, PINNs use MLPs and employ gradient descent optimization algorithms such as Adam and L-BFGS for training the parameters of the Neural Network. Adam and L-BFGS are the algorithms of choice in scientific deep-learning because of their attractive ability to scale efficiently in large numbers of parameters. Replacing them with LM is no so simple, mainly because the LM introduces an important burden per epoch in both calculations and memory consumption. LM needs the maintenance, calculation and inversion of large and dense matrices such as Jacobian and Hessian's approximation. To keep this burden in acceptable levels, the researchers trained using LM only shallow neural networks. MLPs with only 1 hidden layer even if there are simple NN architectures are quite powerful if are trained correctly. The results of this paper show that LM with shallow neural networks consistently outperforms in accuracy by orders of magnitude the pair of Adam+L-BFGS with multilayer neural networks for a specific class of problems.
Finally, the authors express their belief that by using better optimization algorithms along with applying domain decomposition of complex problems into smaller subdomains, Shallow Networks might be able to compete with Deep Networks in representing complex physical phenomena.

4.2.4 Domain Decomposition & XPINNs

XPINNs stand for eXtended Physics-Informed Neural Networks [22]. They represent an extension of the original PINN architecture. One of the key features of XPINNs is their use of domain decomposition. This means the entire space-time continuum is divided into smaller subdomains (Figure 37).

The domain decomposition strategy employed by XPINNs offers several benefits:

1. Parallelization Capacity:

- Each subdomain can potentially be processed independently.
- This allows for efficient parallel computing, which can significantly reduce computation time for large-scale problems.

2. Large Representation Capacity:
    - By dividing the problem into smaller subdomains, XPINNs can handle larger and more complex problems.
    - Each subnetwork can focus on a specific part of the problem, potentially leading to better overall accuracy.
3. Efficient Hyperparameter Tuning:
    - With XPINNs, hyperparameters can be optimized separately for each subdomain.
    - This allows for more tailored solutions to different aspects of the problem.
4. Effectiveness for Multi-scale and Multi-physics Problems:
    - Different subdomains can capture phenomena at various scales or physics simultaneously.
    - This makes XPINNs particularly well-suited for problems involving multiple physical processes or vastly different characteristic lengths/timescales.

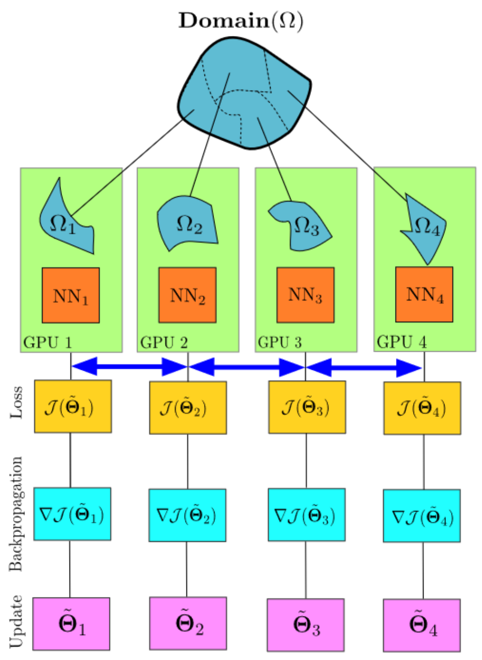

Domain(Ω)
Ω_1
Ω_2
Ω_3
Ω_4
NN_1
NN_2
NN_3
NN_4
GPU 1
GPU 2
GPU 3
GPU 4
Loss
$\mathcal{J}(\tilde{\Theta}_1)$
$\mathcal{J}(\tilde{\Theta}_2)$
$\mathcal{J}(\tilde{\Theta}_3)$
$\mathcal{J}(\tilde{\Theta}_4)$
Backpropagation
$\nabla\mathcal{J}(\tilde{\Theta}_1)$
$\nabla\mathcal{J}(\tilde{\Theta}_2)$
$\nabla\mathcal{J}(\tilde{\Theta}_3)$
$\nabla\mathcal{J}(\tilde{\Theta}_4)$
Update
$\tilde{\Theta}_1$
$\tilde{\Theta}_2$
$\tilde{\Theta}_3$
$\tilde{\Theta}_4$

Figure 37. An illustration of how the domain decomposition works in X-PINNs [22]

5 Conclusions

PINNs is an innovative technology which expands the learning capabilities of standard NNs beyond their current limits. PINNs can approximate physical systems not only by learning from simulation data, as their ancestors, but also by solving directly the underlying system of differential equations.

Our conducted experiments have shown that this technology is competitive in accuracy to standard numerical methods for several benchmark problems. Our PINNs found to have the same or even better accuracy levels compared to other implementations in the literature while using smaller NN architectures and less boundary and collocation points. However, the computational cost of their training introduced mainly by the Automatic Differentiation overhead and its iterative nature make PINNs less efficient and appealing compared to the established numerical solvers. Additionally, PINNs have shown difficulties in finding reliable solutions as the complexity and non-linearity of the problem increases. We showed that using MLPs having the same architecture as PINNs were able to learn the underlying dataset, indicating in this way that the learning weakness of PINNs is not related to the learning capacity of the neural network but instead is consequence of the optimization problem's complexity itself. Trying to locate answers to the endogenous pathologies of PINNs, we studied several research papers, and we presented the most important as candidate solutions for a future state-of-the-art implementation.

Finally, this master thesis is dedicated to those who have the belief that PINNs will play a vital role to the shaping of the next generation neural network based PDE solvers.

6 Future work

The findings of this study highlight several areas where further research could significantly enhance the performance and applicability of Physics-Informed Neural Networks (PINNs). Future work should focus on addressing the computational efficiency issues and improving the reliability of PINNs for complex problems.

1. GPU acceleration: Implementing PINN training on GPU hardware could significantly reduce computation time and enable larger-scale simulations. Optimizing the automatic differentiation process for parallel execution on GPUs would be crucial.

2. Scalable second-order optimization: Developing more efficient second-order optimization algorithms tailored for PINN training could lead to faster convergence and improved accuracy. Adapting existing methods like BFGS or implementing scalable novel variants of Levenberg-Marquardt could be a game changer in PINNs training.

3. Coupling traditional numerical solvers with PINN methodology: Integrating PINN methodology within established numerical methods could create hybrid approaches that leverage the strengths of both paradigms. This could potentially combine the reliability and efficiency of traditional solvers with the flexibility and generalization capabilities of PINNs.

Bibliography – References – Online sources

[1] C. A. Putra, P. Palar, R. Stevenson and L. R. Zuhal, "On Physics-Informed Deep Learning for Solving Navier-Stokes Equations," in *AIAA SCITECH 2022 ForumAt: San Diego, California*, 2022.

[2] I. E. Lagaris, A. Likas and D. I. Fotiadis, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *Physics - Computational Physics,* 1997.

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

[4] "COMPUTER SCIENCE," [Online]. Available: https://computingstudy.wordpress.com/scientific-computing/.

[5] B. Mwandau and M. Nyanchama, "Investigating Keystroke Dynamics as a Two-Factor Biometric Security," 2018.

[6] "Javatpoint," [Online]. Available: https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow.

[7] "pvigier's blog," [Online]. Available: https://pvigier.github.io/2017/07/21/pychain-part1-computational-graphs.html.

[8] "dongminlee.tistory.com," [Online]. Available: https://dongminlee.tistory.com/24.

[9] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2017.

[10] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," in *Mathematical programming*, 1989.

[11] B. Moseley, "BEN MOSELEY. Scientific Machine Learning," [Online]. Available: https://benmoseley.blog/my-research/.

[12] I. d. C. Guerra, W. Li and R. Wang, "A Comprehensive Analysis of PINNs for Power System Transient Stability," *Electronics,* 2024.

[13] S. Markidis, "Physics-Informed Deep-Learning for Scientific Computing," in *arXiv:2103.09655v1*, 2021.

[14] P. Sikdar, S. M. Dash and K. P. Sinhamahapatra, "Lattice Boltzmann Simulations of a Lid-Driven Cavity at Different Moving Lengths of the Top Lid," 2019.

[15] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer and M. W. Mahoney, "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning," 2020.

[16] 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,* 1982.

[17] S. Wang, H. Wang and P. Perdikari, "On the eigenvector bias of fourier feature networks: Fromregression to solving multi-scale PDEs with physics-informed neural networks.," in *Computer Methods in Applied*, 2021.

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

[19] P.-H. Chiu, J. C. Wong, C. Ooi, M. H. Dao and Y.-S. Ong, "CAN-PINN: A Fast Physics-Informed Neural Network Based on Coupled-Automatic-Numerical Differentiation Method," in *arXiv:2110.15832*, 2022.

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

[21] G. K. Yadav and B. Srinivasan, "Shallow Physics Informed Neural Networks Using Levenberg-Marquardt Optimization," in *OPT2020: 12th Annual Workshop on Optimization for Machine Learning*, 2020.

[22] K. Shukla, A. D. Jagtap and G. E. Karniadakis, "Parallel Physics-Informed Neural Networks via Domain Decomposition," *arXiv:2104.10013,* 2021.