Physics-Informed Neural Networks: A Brief Introduction

Group 5: Summer Semester 2023

Abstract: Physics-Informed Neural Networks (PINNs) are gaining popularity as powerful tools for solving partial differential equations (PDEs) of nonlinear systems. PINNS are an algorithm very susceptible to convergence problems that make achieving a correct solution non-trivial, specially in problems of high input dimensionality. The gradient descent algorithm coupled with the weighted objectives method is usually used to optimize loss functions in the PINN training due to the advantage of low computational cost, but since the interaction mechanisms between the gradients of loss functions are not fully clarified, they are leading to poor performances in loss functions optimization such as low efficiency and non-convergence. In this study we demonstrate how these physics informed neural networks can be used for solving incompressible NSE which describe any given law of physics, and obtain physics-informed surrogate models that are fully differentiable with respect to all input coordinates and free parameters. We implemented L-BFGS-B which performs better with a faster rate of convergence and reduced computing cost for cases with a little amount of training data or residual points and then validated by two dimensional incompressible Navier-Stokes equations. By tensorflow we build an auto differentiation class, to stream function and pressure and compute all partial derivatives.

Keywords: Physics-Informed Neural Networks; Partial differential equations; Navier-Stokes equations; Loss functions optimization; L-BFGS-B algorithm.

Contents

1 Introduction **2**

2 Neural Networks: The Basics **3**
2.1 Weights, Biases and Activation Functions . . . 3
2.2 The Concept of "Learning" and Loss Functions . . . 4

3 Neural Networks and PDEs **4**
3.1 Physics-Informed Neural Networks . . . 5

4 PINN Modeling and Testing for 2D Navier-Stokes Equations **5**
4.1 Methodology . . . 6
4.1.1 Architecture of neural network . . . 6
4.1.2 Loss Function . . . 7
4.1.3 Auto Grident . . . 7
4.1.4 Physics Incorporation . . . 7
4.2 Testing for Lid-Driven Cavity Flow . . . 7
4.2.1 Problem Background . . . 7
4.2.2 Testing setting . . . 8
4.2.3 Performance Metrics . . . 8

5 The Curse of Dimensionality in PINNs **8**
5.1 The challenges of high dimensionality on PINNs . . . 8
5.1.1 Computational Complexity . . . 8
5.1.2 Training Complexity . . . 9
5.1.3 Data Scarcity . . . 9
5.1.4 Overfitting Risk . . . 9
5.1.5 Extrapolation Difficulties . . . 9
5.2 Mitigating the Curse of Dimensionality . . . 9
5.2.1 Regularization Techniques . . . 9
5.2.2 Network Architecture Design . . . 10
5.2.3 Data Strategies . . . 10
5.2.4 Dimensionality Reduction . . . 10
5.2.5 Boundary Conditions . . . 10

6 L-BFGS-B algorithm **10**
6.1 Algorithm . . . 10

7 Conclusions **11**

8 Contributions of Authors **12**

1 Introduction

With the considerable growth of available data and computing resources, deep neural network has gradually highlighted its advantages in information extraction for complex nonlinear systems. Nowadays, deep neural network which can extract relevant physical laws from a large number of data samples without any physical information (unsupervised learning), has been used in different fields, including natural language processing, computer vision, machine translation, medical image analysis. However, for some complex engineering problems, the sample data is difficult to obtain. Hence, the purely data-driven deep neural network methods that require a large amount of sample data have great difficulties in dealing with such problems failing to provide the convergence. The principled physical laws that govern the time-dependent dynamics of a system, this information constrains the space of admissible solutions to a manageable size to differentiate neural networks with respect to their input coordinates and model parameters to obtain physics informed neural networks. Fortunately, specific governing equations can describe the physical information involved in complex nonlinear systems. For example, complex flow problems can be characterized by the Navier-Stokes equations. Therefore, incorporating more physical information into a data-driven neural network may provide a new way to solve complex nonlinear systems and this kind of neural network is the Physics-Informed Neural Network (PINN) which is the subject of this study. PINNs are trained by incorporating physical laws as soft constraints in the loss function. The main advantage of a Physics Informed Neural Network is that the physical laws that would constrain the space of solutions are the tool that guides the optimization process towards the desired solution. PINNs using physics prior have the potential to be an effective way of solving high-dimensional PDEs. They have the ability to learn from data and uncover the hidden laws of physics without needing explicit knowledge of the governing equations. This makes them suitable for tackling problems with nonlinear physics, where deriving explicit equations can be difficult or even impossible. Moreover PINNs offer greater flexibility and adaptability compared to traditional solvers. They excel at handling irregular geometries, varying boundary conditions, and learning from limited or noisy data, all without the need for laborious mesh generation or grid discretization. On the other hand, neural network structures based on physics-informed models are significantly slower in training convergence compared to supervised-based networks, specifically if a first order iterative optimization algorithm is used. In particular, they can fail to learn complex physical phenomena, like solutions that exhibit multi-scale, chaotic, or turbulent behavior. Furthermore physics-informed models seek higher memory of Graphics Processing Unit (GPU) compared to supervised models. This is because presence of a more complicated loss function in physics-informed models ends in a more complicated loss function gradient. In the training of PINN, the process of minimizing the loss functions of the neural network is a typical optimization problem. For this optimization, Adam optimizer increase convergence speed, because stochastic gradient descent (SGD) hardly manages random collocation points. But for some specific systems, the convergence difficulty still exists, for example, the Adam optimizer almost fails for the heat conduction equation. So the proposed L-BFGS-B

is introduced to solve two-dimensional incompressible Navier-Stokes equations, to reduce computing cost and for further global optimization.

2 Neural Networks: The Basics

At its core, a neural network (NN) is a computational model designed to recognize patterns. An example that illustrates this is one of the most basic kinds of NNs: a fully connected NN. Its structure is composed of interconnected nodes arranged in layers. Each neuron in a layer is connected to all the neurons in the previous and subsequent layer, with the exception of the input and output layer. Each connection between these neurons has a weight, and each neuron possesses a bias. An example of such a Neural Network can be found in Figure 1 below.

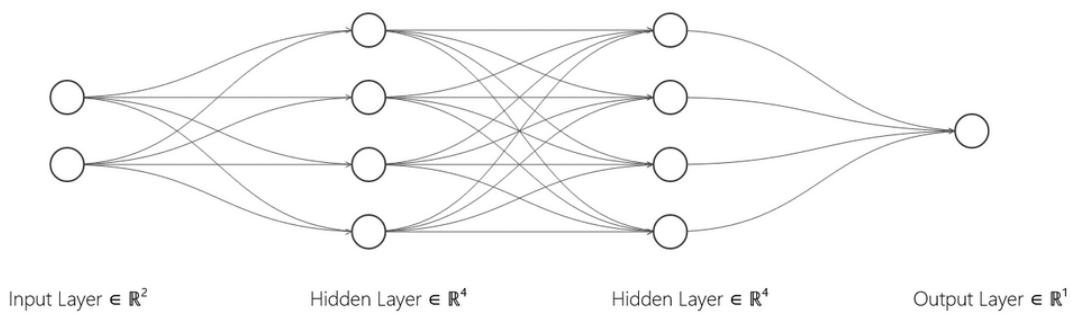

Figure 1: A Visualization of a fully connected NN with 2 input nodes, 2 hidden layers with 4 nodes each and one output node

2.1 Weights, Biases and Activation Functions

The structure of the weights and biases introduced in the paragraph above can be summarized in the following mathematical formula

$$a_j = f(\sum_{i=1}^{I}(\omega_{ij}x_i) + b_j). \tag{1}$$

The value for all nodes, a_j, in each layer $j\epsilon[1, ..., J]$ can be expressed as as the sum of all the weights of the connections to the previous layer ω_{ij} multiplied by the values of the nodes in the previous layer x_i, where $i\epsilon[1, ..., I]$, plus the bias of node a_j have all passed through a so-called activation function $f()$. The activation function $f()$ is used to introduce non-linearity to equation (1). Common activation functions include...The choice for an activation function is problem dependant.
Calculating the values for each layer based on the values of the nodes in the previous layer is called a Forward Pass.

2.2 The Concept of "Learning" and Loss Functions

Training a neural network typically involves feeding it input data and adjusting the weights and biases to minimize the difference between its predicted output and the actual target output. This difference, quantified using a loss function, serves as a measure of how well the network is performing. Trying to minimize this loss function is at the heart of what we would consider learning. In a lot of the most basic neural networks, the loss function can simply be the mean-square-error between the predicted output and the known output.

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

where: y is the true output, $\hat{y}$ is the predicted output and N is the number of outputs. In order to understand how we have to update the weights and biases we have to perform the so called Backwards Pass or Backpropagation.
Without going into too much detail, we can use the chain rule and basic calculus to calculate how our output depends on each of the weights and biases and adjust them accordingly to minimize the loss. Abstractly, we can think of this as gradient descent, where we first calculate our gradient landscape and then take steps towards the minima. A visual description of the process illustrated above can be seen in Figure 2 below.

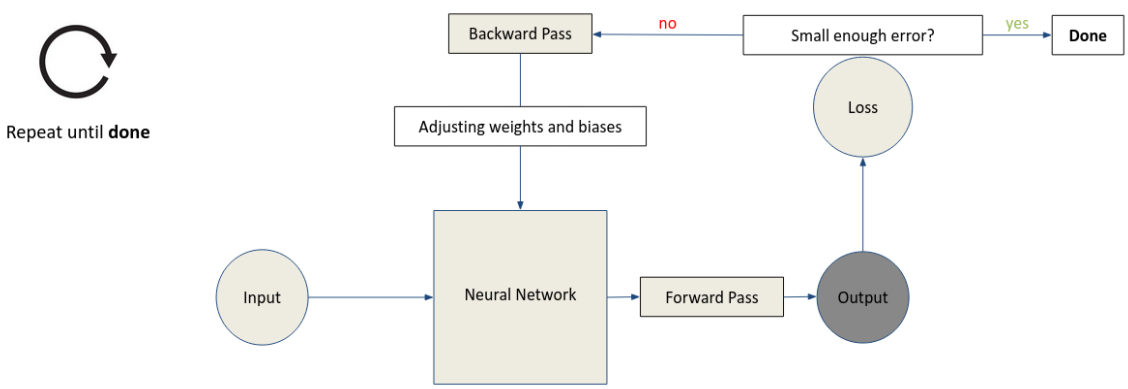

Figure 2: Visual Depiction of the basic learning process of a Neural Network

3 Neural Networks and PDEs

The pillar on which the idea of applying NN to Partial Differential equations (PDEs) stands is the Universal Approximation Theorem. This theorem states that a neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function under certain conditions. With this in mind, we know that in theory, we can approximate any solution to a PDE with arbitrary precision. The difficulty lies in actually finding this solution.

3.1 Physics-Informed Neural Networks

While traditional NNs learn from data, Physics-Informed Neural Networks (PINNs) integrate knowledge of underlying physical laws to inform their learning. The critical difference lies in the loss function:

In addition to the data mismatch term present in regular NNs, PINNs have another term that quantifies the violation of physical laws, usually represented as PDEs. By minimizing this combined loss function, the PINN is simultaneously learning from data and ensuring that its predictions adhere to known physical principles. The data mismatching terms that were mentioned above could be any known points. When it comes to PDEs this could be Boundary Conditions, Initial Conditions or just points inside the domain that we know the solution to.

Assuming a generic partial differential equation of the form:

$$\mathcal{D}(u, u_t, u_{xx}, ...) = 0$$

The total loss function can be expressed as:

$$\mathcal{L} = \mathcal{L}_{BC} + \mathcal{L}_{IC} + \mathcal{L}_P$$

1. Boundary Conditions Loss:

$$\mathcal{L}_{BC} = \sum_i |NN(x_{BC,i}, t_{BC,i}) - u_{BC,i}|^2$$

2. Initial Conditions Loss:

$$\mathcal{L}_{IC} = \sum_i |NN(x_{IC,i}, t_{IC,i}) - u_{IC,i}|^2$$

3. Physics Loss:
where $\mathcal{D}$ is a differential operator, the residual loss can be computed as:

$$\mathcal{L}_R = \sum_i |\mathcal{D}(NN(x_i, t_i), \partial_t NN(x_i, t_i), \partial_{xx} NN(x_i, t_i), ...)|^2$$

where $NN(x, t)$ is the output of the neural network representing the solution. u_{BC} and u_{IC} represent the known boundary and initial values. x_{BC}, t_{BC}, x_{IC} and t_{IC} represent the sets of coordinates of the boundary and initial conditions.

4 PINN Modeling and Testing for 2D Navier-Stokes Equations

The Navier-Stokes equations form the cornerstone of fluid dynamics, describing the behaviour of the fluid flow. In our project, we focused on specific formulations of these

equations, addressing aspects such as incompressible flow. The equations played a pivotal role in defining our modelling objectives.

$$\rho\left(\partial_t \mathbf{v} + (\mathbf{v} \cdot \bigtriangledown)\,\mathbf{v}\right) - \eta \bigtriangleup \mathbf{v} + \bigtriangledown \mathbf{p} = \mathbf{f}$$

in $\Omega \times (0, T)$,

$$div\mathbf{v} = 0$$

in $\Omega \times (0, T)$,

$\mathbf{v} = (u, v)$ is the velocity vector. p is the pressure. ρ is the density and η is the viscosity.

4.1 Methodology

4.1.1 Architecture of neural network

We tried several Physics-Informed Neural network architectures, each with a tailored architecture, and finally made our choice. The networks consisted of multiple layers, incorporating various types of neurons and activation functions. These architectures were optimized to effectively capture the complexities of the Navier-Stokes equations.

time_dependent.py− > InitializeModel

Network setting:

Network: 5 layers, 10 neurons for each layer

Activation function: tanh

Kernel initializer: glorot normal

Regularizer: L2

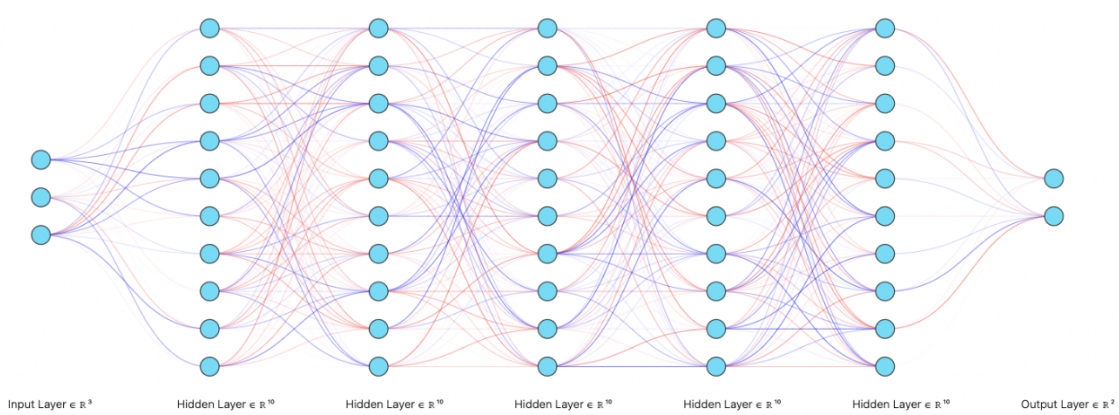

Figure 3: A Visualization of a fully connected NN with 3 input nodes, 5 hidden layers with 10 nodes each and 2 output node

4.1.2 Loss Function

Here, we built two loss functions, one for the Navier Stokes Equations, and one for boundary condition.

$$\mathbf{L}_{boundary} = \frac{1}{\mathbf{N}_{boundary}} \sum_{i=1}^{\mathbf{N}_{boundary}} \left(u\left(x_i\right) - g\left(x_i\right)\right)^2 \tag{3}$$

$$\mathbf{L}_{internal} = \frac{1}{\mathbf{N}_{internal}} \sum_{i=1}^{\mathbf{N}_{internal}} \left(F_x\right)^2 + \left(F_y\right)^2 + \left(C_{free}\right)^2 \tag{4}$$

$$F_x = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2}\right) \tag{5}$$

$$F_y = \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2}\right) \tag{6}$$

$$C_{free} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \tag{7}$$

4.1.3 Auto Grident

In order to make the process more efficient, we build an auto gradient class, detail presented in the time_dependent.py− > classGradientLayer

4.1.4 Physics Incorporation

To enforce the Navier-Stokes equations, we integrated them into the loss function of our neural networks. By giving the default setting

time_dependent.py− > class PINN
inlet flow velocity: u0 = 1
density : rho = 1
viscosity : nu = 0.01
number of training samples: num_train_samples = 1000
number of test samples: num_test_samples = 20

4.2 Testing for Lid-Driven Cavity Flow

4.2.1 Problem Background

Lid-Driven Cavity Flow, a classic problem in fluid dynamics, served as our testing ground. This problem involved a fluid-filled cavity with a moving lid, presenting complex flow patterns. Solving this problem was essential for evaluating the effectiveness of our PINNs in real-world scenarios.

4.2.2 Testing setting

In this scenario, we set the velocity from the downside to u = −1, while v and other boundaries remained at 0. Other settings were in accordance with the parameters defined in our previous neural network configuration.

4.2.3 Performance Metrics

Given the absence of an extensive dataset, direct measurement of the model's performance was challenging. To assess our models, we generated a time series frame to create an animation. This visual representation offers an intuitive way to study simulation performance and gain a deeper understanding of our models' behaviour.

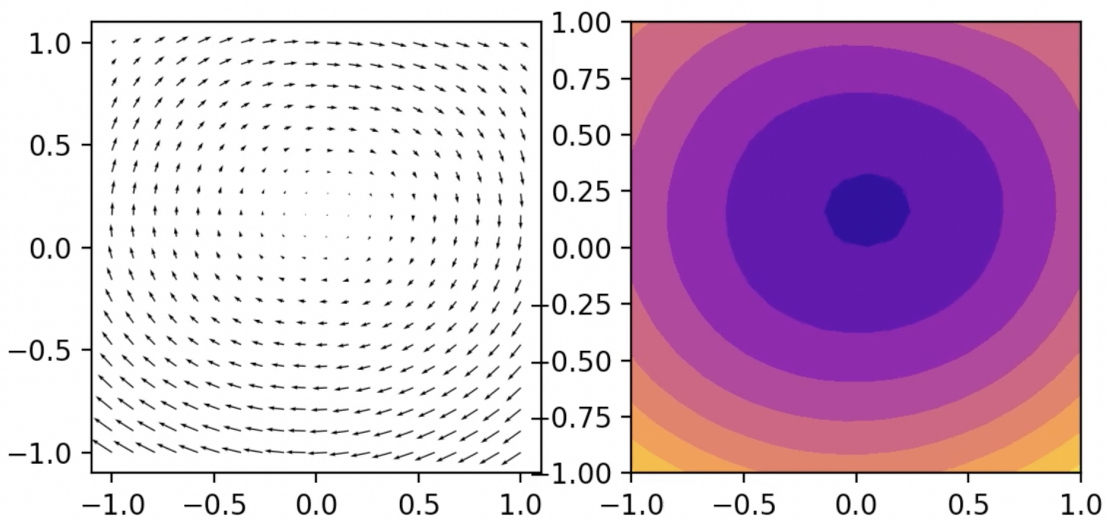

Figure 4: A Visualization of Lid-Driven Cavity Simulation: The left side showcases the velocity field, while the right side displays the pressure field.

5 The Curse of Dimensionality in PINNs

The curse of dimensionality poses significant challenges when applying Physics-Informed Neural Networks (PINNs) to high-dimensional problems. We highlight the challenges posed by high dimensionality and present strategies to overcome these challenges, ensuring that PINNs remain a valuable tool for tackling complex, multi-dimensional physical problems.

5.1 The challenges of high dimensionality on PINNs

5.1.1 Computational Complexity

As the number of dimensions increases, the computational demands of training and evaluating PINNs grow exponentially. The increased dimensionality results in a larger

input space and an abundance of network parameters, leading to substantial computational costs and time requirements. Addressing this issue requires efficient parallelization, hardware acceleration, or dimensionality reduction techniques where applicable.

5.1.2 Training Complexity

Handling high-dimensional systems introduces additional complexities in designing network architectures and formulating suitable loss functions. The dimensionality of the problem space necessitates more intricate network structures and loss functions, which can be challenging to design effectively. Overcoming this complexity involves employing expert domain knowledge and experimenting with a variety of architectures to identify the most suitable one.

5.1.3 Data Scarcity

In high-dimensional problems, acquiring sufficient training data becomes more challenging. Sparse data points are likely, making it difficult for PINNs to accurately capture underlying patterns. This issue can lead to overfitting or poor generalization, limiting the reliability of the model. Strategies for addressing data scarcity include the collection of more diverse and representative data or the use of dimensionality reduction techniques to enhance data density.

5.1.4 Overfitting Risk

The curse of dimensionality amplifies the risk of overfitting in PINNs. With limited data available in high-dimensional spaces, the model may fit noise or specific data points excessively, leading to poor generalization. Effective regularization techniques and rigorous validation strategies are essential to mitigate overfitting risks and ensure model robustness.

5.1.5 Extrapolation Difficulties

PINNs may struggle to accurately extrapolate or predict beyond observed data points in high-dimensional spaces. As dimensionality increases, the model's ability to generalize diminishes, potentially resulting in inaccurate predictions. Overcoming extrapolation difficulties involves careful consideration of boundary conditions and additional training data in unexplored regions of the problem domain.

5.2 Mitigating the Curse of Dimensionality

To address the curse of dimensionality in PINNs, several strategies can be employed:

5.2.1 Regularization Techniques

Regularization methods such as weight decay, dropout, and early stopping can help prevent overfitting in high-dimensional PINNs. Properly calibrated regularization parameters can enhance the model's generalization performance.

5.2.2 Network Architecture Design

Customizing network architectures to the specific characteristics of the high-dimensional problem can significantly improve model performance. Employing domain-specific knowledge in network design is invaluable.

5.2.3 Data Strategies

Efforts to obtain more diverse and representative training data are crucial. Data augmentation, adaptive sampling, or active learning can help increase data density in high-dimensional spaces.

5.2.4 Dimensionality Reduction

n cases where dimensionality reduction is possible without significant loss of information, applying techniques such as Principal Component Analysis (PCA) or autoencoders can simplify the problem and reduce computational complexity.

5.2.5 Boundary Conditions

Explicitly defining boundary conditions and incorporating them into the loss function can enhance the model's ability to extrapolate accurately in high-dimensional spaces.

6 L-BFGS-B algorithm

In this study, an L-BFGS-B algorithm is proposed. The code is easy to implement, and there is no need to supply information about the Hessian matrix or about the structure of the objective function. Moreover, storage requirements of the algorithm are modest and can be controlled by the user, and the cost of the iteration is low and is independent of the properties of the objective function. However, L-BFGS-B on difficult problems can take a large number of function evaluations to converge, and the algorithm cannot make use of knowledge about the structure of the problem to accelerate convergence.

We consider the following quadratic problem over the subspace of free variables:

$$\min\{mk(x) : x_i = x_{ic}, \forall i \in A(x_c)\}$$

subject to $l_i \leq x_i \leq u_i$ for $\forall i \notin A(x_c)$, with the active set $A(x_c)$.

6.1 Algorithm

Choose a starting point x_0 and an integer m that determines the number of limited memory corrections stored. Define the initial limited memory matrix to be the identity and set $k := 0$.

1. If the convergence test $\|P(x_k - g_k, l, u) - x_k\| < 10^{-5}$ is satisfied, stop, where $P(x_k - g_k, l, u) - x_k$ is the projected gradient, g_k is the gradient, and x_k is the current iterate.

2. Compute the Cauchy point x_c by Algorithm CP.

3. Compute a descent direction d_k by either the direct primal method, the conjugate gradient method, or the dual method.

4. Perform a line search along $d_k = \bar{x}_k + 1 - x_k$ subject to the bounds on the problem to compute a steplength and set $x_{k+1} = x_k + d_k$, where $\bar{x}_{k+1}$ is an approximation solution, and x_{k+1} is a new iterate. The line search starts with the unit steplength satisfying the sufficient decrease condition $f(x_{k+1})f(x_k) + \alpha d_k$ with $\alpha = 10^{-4}$ and attempts to satisfy the curvature condition $\|d_k\| \leq \beta\|d_k\|$ with $\beta = 0.9$.

5. Compute $\nabla f(x_{k+1})$.

6. If it satisfies $y_k > \epsilon\|y\|_2$ with a small positive constant $\epsilon = 2.2 \times 10^{-16}$ and a correction pair $\{s_k, y_k\}$, add y_k and s_k to Y_k and S_k. If more than m updates are stored, delete the oldest column from S_k and Y_k, where $s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, with $n \times m$ correction matrices $Y_k = [y_{k-m}, \ldots, y_{k-1}]$, $S_k = [s_{k-m}, \ldots, s_{k-1}]$.

7. Update S_k, Y_k, R_k, and L_k and set y_k/s_k, where α is a positive scaling parameter, and L_k is the $m \times m$ matrix.

8. Set $k := k + 1$ and go to 1.

7 Conclusions

Due to the advantage of low computational cost, the gradient descent algorithm coupled with the weighted objectives method are usually used for loss functions optimization in PINN training. However, the interaction mechanisms between the gradients of loss functions are not fully understood, resulting in low efficiency and non-convergence problems in loss functions optimization. To overcome these problems, an L-BFGS-B algorithm is proposed, and then two-dimensional incompressible Navier-Stokes equations are introduced to validate it. PINNs use a deep neural network to represent the solution of a differential equation, which is trained using a loss function which directly penalises the residual of the underlying equation. Compared to classical numerical methods PINNs have several advantages, for example their ability to provide mesh-free solutions of differential equations and their ability to carry out forward and inverse modelling within the same optimisation problem. The PINN algorithm which is a mesh-free technique, finds PDE solutions by converting the problem of directly solving the governing equations into a loss function optimization problem. So PINNs offer greater flexibility and adaptability exceling at handling irregular geometries,

varying boundary condition , and learning from limited or noisy data, all without the need for laborious mesh generation or grid discretization. However, the distribution of training points influences the flexibility of PINNs. Increasing the number of training points obviously improves the approximation. Moreover, PINNs have a standard setup which remains mostly unchanged except for the formulation of the loss function. They approximate functions and their derivatives nonlinearly and have the ability to describe latent nonlinear state variables that are not observable. For instance, when a variable's measurement was missing, the PINN implementation was capable of accurately predicting that variable. However, the theoretical convergence properties of PINNs are still poorly understood when compared to classical approaches. For example, the PINN loss function can be highly non-convex and result in a stiff optimisation problem, yet it is unclear which classes of problems this affects. Furthermore PINNs struggle to accurately approximate the solution of PDEs when compared to other numerical methods designed for a specific PDE, in particular, they can fail to learn complex physical phenomena. The Navier-Stokes equations contain multi flow parameters and they are intercoupling in the governing equations, resulting in the acceleration effect of the optimizer for the NavierStokes equations to be much less than that for the analytical PDEs. The PINN method coupled with the optimizer might be a potential technique to conduct the time series forecasting for unsteady Navier-Stokes equations and it can be easily extended to deal with other nonlinear systems which are governed by PDEs.

8 Contributions of Authors

1. Kejdi Danglli: Abstract, Introduction, L-BFGS-B algorithm
2. Leon Armbruster: Chapter 2 and 3
3. Haiyun Ning: PINN Modeling and Testing for 2D Navier-Stokes Equations
4. Sajal Kumar Das: The Curse of Dimensionality in PINNs
5. Rinita Shai: Conclusion