POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering

Data Assimilation Based on Physics-Informed Neural Networks for Hemodynamics

Supervisors
Prof. Umberto Morbiducci
Prof. Alessandro Veneziani

Candidate
Micol Bracco

Academic Year 2024 – 2025

Abstract

Traditional approaches for solving partial differential equations (PDEs), including Finite Element, Finite Volume, and Finite Difference methods, face several challenges. These include high sensitivity to uncertainties in boundary conditions, the complexity of generating meshes conforming to the geometry of the domain, and significant difficulties in addressing high-dimensional problems. In this context, Physics-Informed Neural Networks (PINNs) represent an advanced deep learning technique that directly incorporates the physical laws governing a given phenomenon, offering an alternative and flexible approach to solving such equations. This work investigates the use of PINNs for solving both direct and inverse problems, with a particular focus on data assimilation, where observational data are integrated into the modeling process. To validate the proposed method, several test cases were considered using the Navier–Stokes equations in both steady and unsteady forms, across two-dimensional and three-dimensional configurations. In a preliminary phase, simple geometries were used to test the model's effectiveness, performing fine-tuning by varying several network parameters, as well as the number, quality, and placement of the data employed. This was followed by the analysis of more complex geometries resembling anatomical structures, with the future goal of studying coronary hemodynamics under physiologically realistic conditions, using ultrasound data as a reference. The focus was on developing methods that were both efficient and accurate, with particular attention to the study of velocity and pressure fields, as well as the computation of Wall Shear Stress (WSS), a clinically important metric considered a key risk factor for atherosclerosis.

The primary goal of this thesis is to demonstrate that PINNs offer an innovative and viable approach, featuring significant advantages in computational efficiency and cost reduction compared to traditional methods, as well as enhanced capability in handling sparse or noisy data, while also identifying potential weaknesses. The results obtained, compared with analytical solutions and simulations using the Finite Element Method, highlight the potential of this methodology. In particular, PINNs demonstrated superior effectiveness when performing data assimilation by integrating observational data into the solution process, indicating a strong potential for solving inverse problems. Nonetheless, this work represents only an initial step: future developments will involve applying PINNs to real anatomical models, such as blood vessels, to further evaluate their effectiveness in clinically relevant scenarios.

Sommario

I metodi tradizionali per la risoluzione delle equazioni alle derivate parziali (PDE), tra cui i metodi agli elementi finiti, ai volumi finiti e alle differenze finite, presentano diverse criticità. Tra queste vi sono l'alta sensibilità alle incertezze nelle condizioni al contorno, la complessità nella generazione di mesh che si adattino alla geometria del dominio e le notevoli difficoltà nel trattare problemi ad alta dimensionalità. In questo contesto, le Physics-Informed Neural Networks (PINNs) rappresentano una tecnica avanzata di deep learning che incorpora direttamente le leggi fisiche che governano un fenomeno, offrendo un approccio alternativo e flessibile alla risoluzione di tali equazioni.

Questo lavoro si propone di indagare l'uso delle PINNs per risolvere sia problemi diretti che inversi, con un'attenzione particolare all'assimilazione dei dati, ovvero all'integrazione di dati osservativi all'interno del processo di modellazione. Per validare il metodo proposto, sono stati considerati diversi casi di studio basati sulle equazioni di Navier–Stokes, sia nella forma stazionaria che non stazionaria, in configurazioni bidimensionali e tridimensionali.

In una fase preliminare, sono state utilizzate geometrie semplici per testare l'efficacia del modello, effettuando un fine-tuning variando diversi parametri della rete, nonché il numero, la qualità e la distribuzione dei dati impiegati. Successivamente, si è passati all'analisi di geometrie più complesse, che richiamano strutture anatomiche, con l'obiettivo futuro di studiare l'emodinamica coronarica in condizioni fisiologicamente realistiche, utilizzando dati provenienti da ultrasuoni come riferimento. L'attenzione è stata posta sullo sviluppo di metodi efficienti e precisi, con particolare riguardo allo studio dei campi di velocità e pressione, nonché al calcolo del Wall Shear Stress (WSS), una grandezza clinicamente rilevante considerata un importante fattore di rischio per l'aterosclerosi.

L'obiettivo principale di questa tesi è dimostrare che le PINNs rappresentano un approccio innovativo e valido, caratterizzato da vantaggi significativi in termini di efficienza computazionale e riduzione dei costi rispetto ai metodi tradizionali, oltre a una maggiore capacità di gestire dati scarsi o rumorosi, pur identificandone anche eventuali punti di debolezza. I risultati ottenuti, confrontati con soluzioni analitiche e simulazioni realizzate mediante il metodo agli elementi finiti, mettono in evidenza il potenziale di questa metodologia. In particolare, le PINNs hanno mostrato una maggiore efficacia nell'eseguire l'assimilazione dei dati, integrando le osservazioni direttamente nel processo di soluzione, evidenziando un forte potenziale per la risoluzione di problemi inversi.

Tuttavia, questo lavoro rappresenta solo un primo passo: sviluppi futuri prevedono l'applicazione delle PINNs a modelli anatomici reali, come i vasi sanguigni, per valutare ulteriormente la loro efficacia in scenari di rilevanza clinica.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cardiovascular diseases (CVDs) represent the leading cause of mortality worldwide. Among the various types of CVDs, coronary artery disease (CAD) is one of the most prevalent and life-threatening conditions. It occurs when one or more coronary arteries become partially or totally obstructed, typically due to the accumulation of atherosclerotic plaques composed of lipids, inflammatory cells, and abnormal calcifications. This progressive narrowing can severely reduce blood flow to the myocardium, potentially triggering acute clinical events such as myocardial infarction.
The risk factors contributing to CAD are multifactorial and include genetic predisposition, lifestyle factors such as smoking, diet, and physical inactivity, and hemodynamic forces. Among these, biomechanical factors, particularly those related to blood flow dynamics, provide critical insight into the early detection and progression of vascular disease.
From a mathematical perspective, blood flow in the coronary arteries is described by the incompressible Navier-Stokes equations for a Newtonian fluid under laminar flow conditions, expressed as [1]:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \nabla \cdot \left[\nu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T \right) \right] + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{f}, \\ \nabla \cdot \mathbf{u} = 0, \end{cases} \tag{1.1}$$

where ρ denotes the fluid density, μ the dynamic viscosity, $\mathbf{u} = (u_x, u_y, u_z)$ the velocity field, and p the pressure. The first equation represents the conservation of momentum, while the second corresponds to the conservation of mass, commonly referred to as the continuity equation.
The analysis of hemodynamic quantities, such as velocity, pressure, and particularly Wall Shear Stress (WSS), provides valuable insights and is therefore essential for improving the diagnosis, prognosis, and treatment planning of coronary artery disease. WSS is defined as the tangential component of the stress tensor $\boldsymbol{\tau}$ exerted by blood on the arterial wall, given by [2]:

$$\text{WSS} \equiv \boldsymbol{\tau} \cdot \mathbf{n} - (\mathbf{n} \cdot \boldsymbol{\tau} \cdot \mathbf{n})\mathbf{n}$$

where $\mathbf{n}$ is the unit normal vector to the arterial wall.

WSS is widely recognized as an early indicator of the onset and development of coronary artery disease. Abnormal patterns of WSS, including low or oscillatory shear, have been strongly correlated with regions susceptible to plaque development and rupture [3]. Since WSS depends directly on the spatial gradient of the velocity field, its precise estimation requires high-resolution spatial and temporal velocity data. Traditional numerical methods may encounter challenges such as high computational cost and difficulties in handling boundary conditions. To overcome these limitations, a more robust approach involves assimilating velocity data into physical-mathematical models, such as the Navier–Stokes equations, thereby obtaining a regularized and physically consistent estimation of WSS.
Among emerging methodologies, Physics-Informed Neural Networks (PINNs) have shown great promise for integrating experimental data with physical constraints. PINNs combine deep learning architectures with the governing equations of physical systems, by embedding the differential operators directly into the loss function used during training. This approach allows the neural network not only to fit available data but also to respect the underlying physics of the problem.
The objective of this thesis is to develop and assess a methodology based on PINNs for solving the Navier–Stokes equations across various problems and for accurately estimating WSS from predicted velocity fields. This represents a first step toward the ultimate goal of contributing to the development of a fast, personalized, and reliable hemodynamic modeling framework to support diagnosis and therapeutic planning in coronary artery disease [4].

The thesis is organized into chapters, sections, and subsections. Chapter 2 reviews various modeling approaches, including Computational Fluid Dynamics (CFD) and alternative methods such as Reduced Order Models (ROMs), Physics-Informed Neural Networks (PINNs), and data assimilation techniques. In Chapter 3, the fundamentals of deep learning are recalled, covering neural network architectures, activation functions, backpropagation, loss functions, and optimization algorithms. Chapter 4 describes the core principles of PINNs. Chapter 5 provides a brief introduction to the mathematical model used. Chapter 6 analyzes the clinical relevance of the WSS in relation to vascular pathologies. Chapters 7 and 8 apply PINNs to two- and three-dimensional geometries, respectively, with Chapter 7 focusing on the Kim&Moin and Womersley problems in two dimensions, and Chapter 8 extending the application to three-dimensional Poiseuille flow. Chapter 9 considers a curved tube geometry to simulate more realistic vascular conditions. Finally, Chapter 10 summarizes the main findings, highlighting the strengths and limitations of the PINN methodology and outlining potential future developments. The Appendix provides a hands-on guide to implementing PINNs using the DeepXDE library.

Chapter 2

Introduction to Modeling Approaches

2.1 Recap of Computational Fluid Dynamics (CFD) Methods

CFD simulation represents the gold standard for solving the equations that describe fluid motion, employing mathematical models to generate a numerical solution. Several numerical methods are available to solve these equations, allowing the continuous domain to be discretized into a numerical representation, typically achieved through a mesh. This process results in a system of algebraic equations that can then be solved numerically [5]. The main methods used for this purpose include FDM (Finite Difference Method), FVM (Finite Volume Method), and FEM (Finite Element Method) [6].
However, despite significant advancements in recent years driven by the development of new tools, the CFD approach continues to face inherent challenges and limitations. These issues become more evident as the complexity of the problem increases, including difficulties in mesh generation, especially in complex geometries, high computational costs, and numerical instability [7, 8]. Moreover, these methods are particularly sensitive to boundary and initial conditions imposed [9]. In consideration of these limitations, alternative methodologies have recently been investigated with the aim of maintaining high accuracy while reducing computational costs. This issue is particularly relevant in the medical field, especially in the cardiovascular context, where blood vessels have complex geometries. Specifically, since Computational Fluid Dynamics is sensitive to uncertainties in image segmentation, boundary conditions, and blood rheology [10], exploring a wide range of flow conditions or complex geometric configurations in three-dimensional models may be computationally expensive and demanding to solve [11].
In this context, there is a growing need to develop alternative solutions, which require further investigation, that allow the acquisition of cardiovascular flow fields at low economic and computational cost, while maintaining adequate accuracy.

2.2 Data Assimilation (DA)

Data Assimilation (DA) refers to a set of techniques that enhance the estimation of a system's true state by systematically combining observational data with computational model outputs [12]. By leveraging both empirical measurements and mathematical models, DA minimizes discrepancies between modeled and observed behaviors, thereby improving the reliability and accuracy of predictions [13].

Physics-based mathematical models used to describe complex systems often present several limitations, such as difficulties in defining boundary conditions or accurately identifying key system parameters. These limitations can undermine the reliability and predictive capability of the model, especially in complex or poorly observable scenarios. To overcome these challenges, data-driven approaches have been developed to integrate observed data within the physical modeling framework, exploiting available empirical information to enhance the representation of the phenomenon under study. Physical modeling and data-driven modeling can be used independently, but they are particularly effective when combined into a hybrid formulation [14, 15].

In clinical contexts, this hybrid approach proves especially valuable, as data are often scarce or uncertain, particularly concerning initial or boundary conditions. Integrating experimental data helps bridge these information gaps and enhances the model's predictive performance [16]. Moreover, the capability to dynamically update the model with new measurements makes these methods adaptable and potentially suitable for real-time applications in diagnostics and therapeutic decision support.

2.3 Alternative Methods

2.3.1 Reduced Order Models (ROMs)

Reduced Order Models (ROMs) constitute an effective strategy for reducing the computational cost associated with Computational Fluid Dynamics (CFD) simulations. The development of these models generally proceeds through two distinct phases: an offline phase and an online phase. During the offline phase, a high-fidelity simulation, referred to as the Full Order Model (FOM), is executed. FOMs are solved using classical numerical methods such as the Finite Element Method (FEM), Finite Volume Method (FVM), or Finite Difference Method (FDM) [17]. This approach is particularly costly but extremely accurate.

During this phase, techniques such as Proper Orthogonal Decomposition (POD) are employed to extract the main features of the high-fidelity model. A fundamental step in this process is the parametrization of the governing equations, meaning they depend on a set of parameters. Sample solutions, or snapshots, are collected by solving the FOM for different values of these parameters.

The collected snapshots are organized as columns within a matrix, to which Singular Value Decomposition (SVD) is then applied. Since many snapshots

may be redundant, the method reduces the model by extracting the eigenvectors corresponding to the largest singular values, as they contain the most significant information in the dataset. These eigenvectors form an orthogonal basis onto which the original high-dimensional problem is projected [18].
Subsequently, during the online phase, the reduced-order problem, characterized by a substantially lower dimensionality, is solved, enabling fast approximations for new parameter values at a fraction of the computational cost.
In this phase, solutions to new scenarios can be predicted by modifying the parameters, while incurring a significantly lower computational cost compared to the Full Order Model [19].
In biomedical applications, this approach is adopted to simulate blood flow within blood vessels. In this context, starting from a database of representative solutions obtained from the FOM, the POD technique is applied to derive the ROMs, which use data from previous simulations as basis functions to quickly solve similar new scenarios [11]. Furthermore, the integration of data assimilation has significantly enhanced predictive accuracy [20]. However, the ROM method also has some limitations. The main challenges involve using ROMs for turbulent flows and for flows that include geometrical parametrization [21].

2.3.2 Physics Informed Neural Networks (PINNs)

Another approach involves the use of deep learning algorithms, specifically Physics-Informed Neural Networks (PINNs). These networks are designed to integrate the physical laws governing the problem directly into the model, such as the Navier-Stokes equations and boundary conditions, directly into the model by incorporating them as loss functions during the training process [22]. Moreover, through data assimilation, available experimental data can also be incorporated into the model [23]. This enables the network to learn solutions that not only satisfy the underlying physical laws but also align with real-world measurements. This method, which constitutes the main focus of this thesis, will be examined in greater detail in the following chapters, with particular attention to its applications in a clinical context.

Chapter 3

An Introduction to Deep Learning

3.1 Architecture

An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the biological nervous system. The human brain is composed of a network of interconnected neurons, which are the fundamental units of the nervous system. Each neuron receives signals through dendrites, processes them within the soma, which contains the nucleus, and transmits the output through the axon to other neurons via synapses [24].

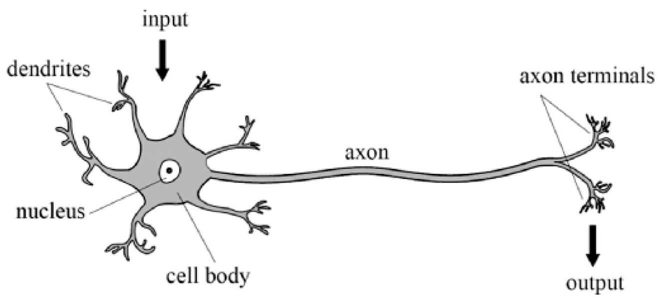


Figure 3.1: Structure of a Biological Neuron [25]

Similarly, an artificial neural network consists of computational units known as artificial neurons or nodes, which represent the fundamental building blocks of the model. Each artificial neuron acts as an input/output processing unit: it receives input signals, computes a weighted sum, and then applies a linear or non-linear transformation known as the activation function, which converts this value into an output signal. These neurons are organized into layers: the input layer, whose size corresponds to the number of input variables; one or more hidden layers, where the main processing of learning takes place; and the output layer, which provides the final predicted values [26].
In the human brain, information transfer and output processing result from the combined activation and inhibition of synapses. In ANNs, the learning process consists of optimizing the synaptic weights, the coefficients that determine the strength of connections between neurons, by minimizing a cost function that measures the difference between predicted and actual values. The ability of a

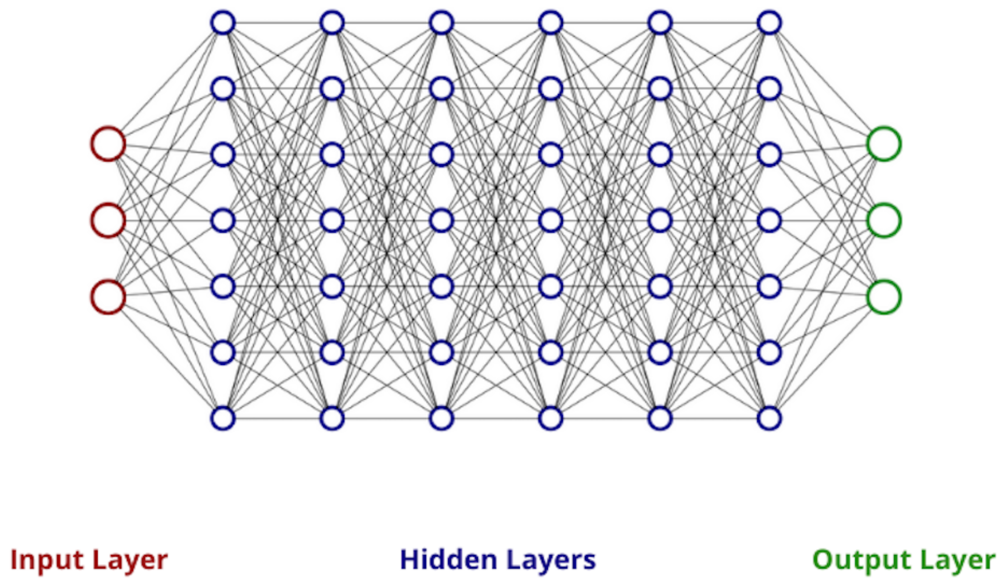


Figure 3.2: General Architecture of an Artificial Neural Network.

neural network to approximate complex functions depends on several key factors: its depth, defined by the number of layers; its width, determined by the number of neurons per layer; the quality and representativeness of the training data; and its capacity to generalize to unseen inputs. Increasing the number of hidden layers transforms the model into a deep neural network (DNN), which enhances its ability to capture intricate and hierarchical patterns in the data. Consequently, the architectural design of the network must be carefully adapted to the nature of the problem and the complexity of the relationships among variables [27].
Depending on the problem to be solved, different types of neural networks can be used, distinguished by how the neurons in different layers are connected and how the layers are organized. A network where each node receives input only from the neurons in the preceding layers and sends its output to the neurons in the following layers is called a feedforward neural network (FNN). If each neuron in one layer is connected to every neuron in the next layer, the feedforward network is called fully connected (FFNN). On the other hand, if some neurons send their output to other neurons within the same layer or to neurons in previous layers, the network is known as a recurrent neural network (RNN) [28].

3.2 Activation Function

Activation functions play a fundamental role in the learning process of neural networks, as they introduce the non-linearity necessary for the network to learn and represent complex patterns. In the absence of such functions, the network would only be capable of modeling linear relationships between input and output, solving only simple problems. However, most real-world data is characterized by non-linear relationships; therefore, activation functions are designed to be non-linear in order to enable the network to learn these complex patterns [29]. Mathematically, the activation function determines the output of a neuron by processing the weighted sum of the inputs x_j, where j refers to the index of the inputs connected to the i-th neuron, multiplied by their corresponding weights w_{ij}, and adding the bias term β, as shown in the expression (3.1) . The weights can be interpreted as the neuron's sensitivity to each input, while the bias β represents the neuron's overall sensitivity or tendency to activate [30].

$$y_i = \sum_j w_{ij} x_j + \beta \tag{3.1}$$

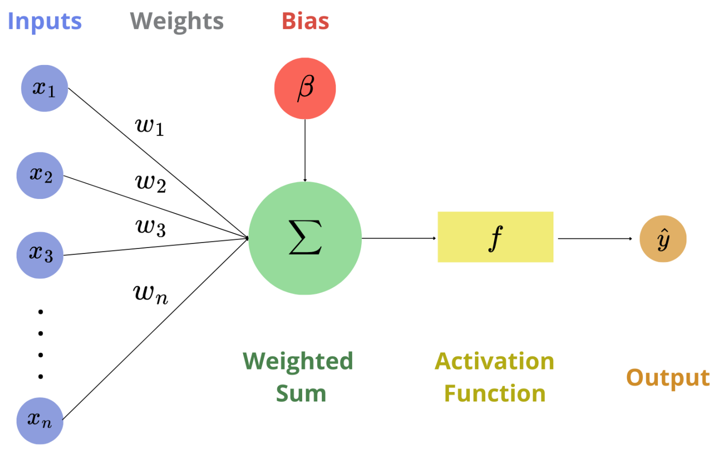


Figure 3.3: Behavior of the activation function: the weighted sum of inputs, summed to the bias, is passed through an activation function f to obtain the output.

3.3 Backpropagation

Backpropagation is the fundamental algorithm that enables neural networks to learn. It computes the gradient of the error function with respect to each weight, indicating how changes in each weight impact the overall error. The learning process starts with forward propagation: data flows through the network layer by layer. Each neuron computes a weighted sum of its inputs, applies an activation function, and produces an output that is passed to the next layer [31]. Initially, the output computed is not accurate. Therefore, after the forward pass, the next step is error calculation, where the difference between the predicted output and the true value is measured using the loss function. This error is then propagated backward through the network in the backpropagation phase. During this step, the algorithm determines how much each weight contributes to the error, allowing the network to update each weight accordingly. The network repeats this cycle — forward propagation, error calculation, backpropagation, and weight updates

— continuously refining the parameters until the error is sufficiently small and the predictions become accurate [32].
The main activation functions used in deep learning are presented below, with particular emphasis on those most commonly employed in PINNs, where the choice of activation function can significantly influence the network's ability to satisfy physical constraints and accurately learn the solution to differential equations.

3.3.1 Logistic Sigmoid

The sigmoid function is an S-shaped curve that outputs values between 0 and 1. Since the function maps inputs to a range between 0 and 1, the outputs can be promptly interpreted as probabilities. This makes it particularly useful for binary classification problems. By compressing the output to a value between 0 and 1, the sigmoid activation function enables the result to be interpreted as a probability, making it particularly suitable for binary classification tasks where the output represents the likelihood that the input belongs to a specific class.
It is defined as follows:

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

The derivative of the sigmoid function can be expressed as:

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} \tag{3.3}$$

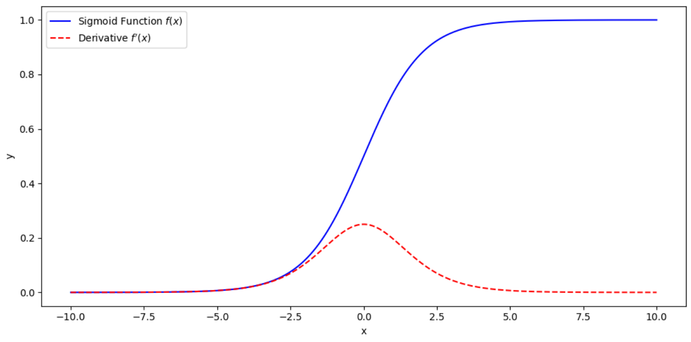


Figure 3.4: Plot of the Sigmoid activation function (in blue) and its derivative (in red).

A critical issue arises when computing the derivatives of activation functions: when neuron inputs are very small or very large, especially in deep networks, the gradients tend to vanish progressively across layers, which significantly impedes the training process. This phenomenon is commonly referred to as the "vanishing gradient problem" [33]. Specifically, if the gradients are already minimal in the earlier layers due to activation saturation, they become even smaller in the deeper layers. Consequently, the weight updates in these deeper layers are almost zero, preventing the effective adjustment of model parameters. This slow adjustment process impedes learning and slows down the overall training of the model [30]. Another issue is that the outputs of the sigmoid function are not zero-centered, meaning they always produce positive values between 0 and 1. This shifts the average value of the inputs to the next layer away from zero, which can disrupt the balance of the data flowing through the network and slow down the convergence during training [34].

3.3.2 Tanh

The hyperbolic tangent function (tanh) maps the inputs in the range between -1 and 1, and it is defined as follows:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.4}$$

The derivative of the hyperbolic tangent activation function can be expressed as:

$$f'(x) = 1 - \tanh^2(x) \tag{3.5}$$

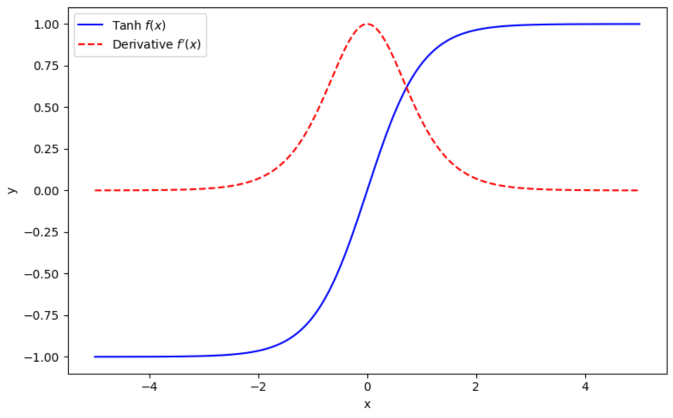


Figure 3.5: Plot of the Tanh activation function (in blue) and its derivative (in red).

During backpropagation, the derivative of the activation function is essential for updating the network's weights. Compared to the sigmoid function, the tanh function outputs values in the range -1 to 1, which not only provides a higher output range but also makes it zero-centered. This zero-centering facilitates more effective weight updates and helps the model converge faster, resulting in quicker training. Additionally, because tanh outputs are balanced around zero, it works particularly well with normalized data, simplifying the optimization process.
However, like the sigmoid function, the tanh function suffers from the vanishing gradient problem.

3.3.3 ReLU

The Rectified Linear Unit (ReLU) function outputs zero for every input less than or equal to zero, and for every input greater than zero, it outputs the input value itself:
The ReLU activation function is defined as:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

The derivative of the ReLU function can be expressed as:

$$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

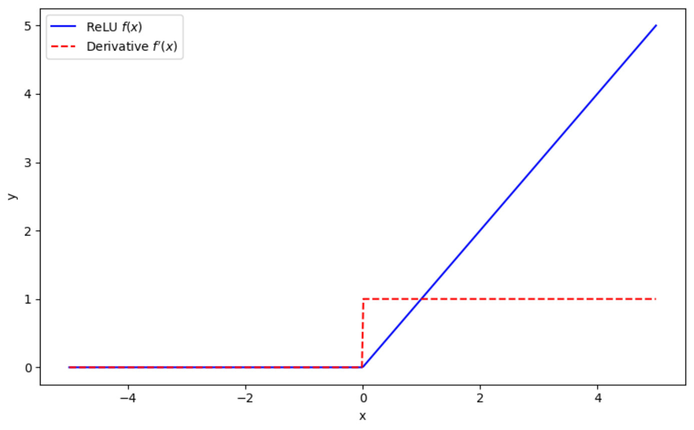


Figure 3.6: Plot of the ReLU activation function (in blue) and its derivative (in red).

The derivative of the ReLU function is piecewise constant and equal to 1 for all positive values, and zero for non-positive values. This characteristic helps mitigate the vanishing gradient problem because the gradients are not diminished when passing through positive values, leading to faster convergence during training. One advantage of the ReLU function is that it behaves linearly for inputs greater than zero, while outputting zero for inputs less than or equal to zero, introducing non-linearity to the network. However, ReLU can suffer from a related issue known as the "dying ReLU problem," a variant of the vanishing gradient problem. This occurs when a large number of ReLU neurons become inactive, outputting zero, during much of the training process. This can prevent the network from learning complex patterns effectively, as the gradients for these neurons are zero, leading to ineffective weight updates [35].

3.4 Loss Function

One of the key parameters in deep learning is the loss function, which measures how well the model approximates the correct output. The goal is to find the point where the function reaches its minimum. Depending on the architecture and the specific problem being addressed, different loss functions can be chosen, as they can significantly impact the performance of the trained network. In general, loss functions can be categorized based on the type of problem being solved: regression or binary classification. Each category includes different types of loss functions suited for their respective tasks [36].

3.4.1 Regression

The Regression method is a learning problem in machine learning that aims to predict a continuous output value based on one or more input features. In this case, the most commonly used loss functions are Mean Squared Error (MSE) and Root Mean Squared Error (RMSE).

Mean Squared Error (MSE)

The Mean Squared Error (MSE) calculates the average of the squared differences between the predicted values by the model and the actual values. Mathematically, the MSE loss function is expressed as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{3.6}$$

where N is the number of samples, $\hat{y}_i$ is the true value of the i-th sample, and y_i is the predicted value of the i-th sample.
MSE, being a quadratic function, promotes efficient convergence toward minima in the presence of small errors, as its gradient gradually decreases. Additionally, it is differentiable, making it particularly suitable for efficient gradient computation, a key element in derivative-based optimization algorithms such as gradient descent. Furthermore, the convexity of the function simplifies the optimization process, allowing gradient-based techniques to converge to the global minimum without getting trapped in local minima [36]. However, one of its disadvantages is that, due to its quadratic nature, MSE tends to amplify the impact of outliers [37].

Root Mean Squared Error (RMSE)

The RMSE is the square root of the mean squared error (MSE) defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{3.7}$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, and N is the number of samples.
The RMSE has several advantages, including being easy to compute and differentiable. As a linear function, it provides a steeper gradient near the minima, allowing for faster adjustments and potentially improving optimization. Unlike MSE, RMSE penalizes large errors less due to the square root, which prevents the scale from being overly amplified. However, RMSE increases proportionally with large errors, making it more sensitive to outliers. This increased sensitivity can lead to higher computational costs and make training neural networks more challenging [37].

3.4.2 Binary Classification

The Binary Classificationis a learning problem in machine learning that trains a model with the goal of predicting the category or class of a given input, aiming to classify the input features into a specific class. Classification problems include binary classification, where the output is one of two possible classes; multi-class classification, involving multiple mutually exclusive classes; and multi-label classification, where instances can be assigned to multiple classes simultaneously.

Binary Cross-Entropy (BCE)

The Binary Cross-Entropy (BCE) loss function measures the difference between the predicted probability of a class, typically labeled as 1, and the actual class label, typically labeled as 0. When the predicted probability p aligns with the true class label y, the BCE loss is minimized, and the model learns effectively. Mathematically, the BCE loss is defined as:

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{3.8}$$

A key advantage of Binary Cross-Entropy lies in its computational efficiency, differentiability for $p > 0$, and the ability to interpret the model's output as a probability distribution. Furthermore, BCE is less influenced by outliers. However, it can encounter challenges with class imbalance, where one class has significantly more samples than the other, potentially leading to biased predictions [38].

3.5 Optimization Algorithm

The optimizer is a fundamental component in the fine-tuning of neural networks, responsible for minimizing the loss function by iteratively adjusting the model parameters in a manner that progressively enhances the model's performance. This optimization process relies on backpropagation, which computes the gradient of the loss function with respect to each model parameter, providing the necessary information for weight updates, and the optimizer uses this information to determine the direction in which the weights should be adjusted to minimize the error. The optimization process is governed by several hyperparameters, including the *learning rate*, the *initialization of weights*, and the number of *training epochs*, all of which must be carefully selected and adjusted to the specific characteristics of the problem under investigation. Typically, the model is trained for a fixed number of epochs or until the loss function reaches a sufficiently low value. When further improvements in performance become negligible and the loss function stabilizes, the model is considered to have reached convergence [39].

Learning Rate

The learning rate controls the step size of the optimization algorithm. The step size determines how big or small each update to the model's parameters will be during training. This parameter affects how quickly or slowly the optimizer moves toward the minimum of the loss function. A large learning rate can lead to rapid convergence but carries the risk of overshooting or getting stuck in a suboptimal local minimum. Conversely, a very small learning rate ensures more precise convergence toward the true minimum; however, it requires a significantly larger number of epochs and increases the risk of becoming trapped in local minima or saddle points [40].
There are different strategies for setting the learning rate. The simplest approach is to use a fixed learning rate throughout the training process. However, this often results in slow convergence and prolonged training times. An alternative approach is to employ a decaying learning rate, where training begins with a relatively high learning rate that gradually decreases over time, typically following an exponential or polynomial decay function. Another widely used strategy is the use of cyclic learning rates, where the learning rate oscillates between predefined bounds following a cyclical pattern [41].

Weights

The weights are parameters that define the strength of connections between neurons in a neural network. Each weight determines how much the output of one neuron influences the input of another. After backpropagation calculates the error gradients associated with each weight, optimizers use these gradients to update the weights. Specifically, optimizers decide both how much and in which direction to adjust the weights, aiming to reduce the overall error. These updates take into account the learning rate, which controls the step size of each adjustment, and the gradient direction indicated by backpropagation [40].

3.5.1 Gradient Descent (GD)

Gradient Descent is an optimization algorithm designed to minimize a function by iteratively updating its parameters. The core idea behind the process is that the gradient of the loss function indicates the direction in which the function increases the most. The algorithm then takes a step in the opposite direction of the gradient, moving toward the point where the loss is minimized [42, 43]. At each iteration, the model's weights are updated according to the following formula, gradually reducing the value of the function:

$$w_{k+1} = w_k - \eta \frac{dL}{dw_k} \tag{3.9}$$

where w_{k+1} represents the new weight, w_k represents the previous weight, η represents the learning rate, and $\frac{dL}{dw_k}$ is the partial derivative of the loss function with respect to the weight w_k, calculated over the entire dataset.
Although it is simple to implement, Gradient Descent does not work for all functions. There are two fundamental requirements for its proper functioning: the differentiability and convexity of the loss function. Differentiability is essential to ensure that the gradient can be calculated continuously and precisely at every point in the domain of the function. Convexity ensures that the function has a single optimal solution, a global minimum, avoiding getting stuck in local minima [44].

3.5.2 Stochastic Gradient Descent (SGD)

Vanilla Stochastic Gradient Descent

One of the main limitations of gradient descent is that, since the gradients are computed over the entire dataset, the algorithm struggles to converge to a local minimum efficiently, requiring a significant amount of time. To address this issue, the Stochastic Gradient Descent (SGD) algorithm was introduced. In SGD, after the backpropagation phase, the optimizer updates the weights using a single data point at a time, randomly selecting training samples from a larger dataset rather than computing gradients over the entire dataset. Since weight adjustments are made incrementally, sample by sample, rather than after processing the full dataset, SGD reduces the number of iterations needed and decreases the computational time required to process large amounts of data [45]. Additionally, the noisy updates in SGD, caused by its stochastic nature, can be both beneficial and detrimental. On one hand, the variability introduced can help the model escape local minima or saddle points, potentially leading to better solutions. On the other hand, these updates may cause oscillations in the loss function, preventing it from decreasing smoothly and impeding stable convergence [46].
In the simplest case, known as Vanilla SGD, the gradients are multiplied by the learning rate and then subtracted from the model's weights to update them:

$$w_{k+1} = w_k - \eta \frac{dL_i}{dw_k} \tag{3.10}$$

where $\frac{dL_i}{dw_k}$ is the partial derivative of the loss function with respect to the weight w_k, calculated on the single sample i.

Stochastic Gradient Descent with Momentum (SGDM)

The Stochastic Gradient Descent method presents some intrinsic limitations, including slow convergence and pronounced oscillations in the weight update trajectories. To address these issues, an extended version of stochastic gradient descent known as SGD with Momentum (SGDM) has been introduced. This approach has proven effective in accelerating convergence and reducing oscillations during the optimization process. Specifically, the weight update is modified according to the following expression:

$$w_{k+1} = w_k - \eta \frac{dL_i}{dw_k} + p \frac{dL_i}{dw_{k-1}} \tag{3.11}$$

where p is the momentum coefficient, with $0 \leq p < 1$, and $\frac{dL_i}{dw_{k-1}}$ is the gradient of the loss function at iteration $k-1$.
In this scheme, the current update incorporates both the information from the current gradient and the update direction from the previous step. Thanks to the addition of the momentum term, when the gradients point in the same direction, their effect is amplified, facilitating convergence; conversely, when the gradients change direction and oscillate, the effect of the momentum term is reduced, contributing to a more stable update [47].
Although the SGDM method often outperforms classical SGD, further improvements have been achieved through appropriate parameter management strategies. A widely adopted strategy in practice is Multistage Stochastic Gradient Descent with Momentum (Multistage SGDM). In this approach, each stage is defined by three key parameters: the learning rate, the momentum coefficient, and the stage length. These parameters are systematically updated from one stage to the next to progressively refine the learning process [48]. This multistage strategy has shown high effectiveness in training large-scale neural networks, where carefully tuning these parameters across stages is essential for achieving optimal performance [49].

3.5.3 Root Mean Square Propagation (RMSProp)

Root Mean Square Propagation (RMSProp) is an optimization algorithm that adaptively adjusts the learning rate by computing an exponential moving average of the squared gradients, thereby enhancing stability and efficiency during optimization [50].
Specifically, at step k, RMSProp updates the exponential moving average of the squared gradients g_k as:

$$E[g^2]_k = \beta E[g^2]_{k-1} + (1-\beta)\left(\frac{\partial L}{\partial w_{k-1}}\right)^2, \tag{3.12}$$

where $\beta \in [0, 1)$ is the decay rate controlling the influence of past gradients.
The parameter update rule is:

$$w_k = w_{k-1} - \frac{\eta}{\sqrt{E[g^2]_k + \epsilon}} \left(\frac{\partial L}{\partial w_{k-1}} \right), \tag{3.13}$$

where η is the learning rate and ϵ is a small constant added for numerical stability. This mechanism scales the updates based on the magnitude of recent gradients, improving convergence and robustness of the algorithm [51].

3.5.4 Adaptive Moment Estimation (ADAM)

The Adaptive Moment Estimation (ADAM) is a stochastic optimization algorithm that serves as an advanced alternative to the classical Stochastic Gradient Descent. Unlike SGD, Adam assigns an adaptive learning rate to each parameter, updating them individually throughout the optimization process. The algorithm combines the strengths of two previously introduced optimization methods: Stochastic Gradient Descent with Momentum (SGDM) and RMSProp [52]. Adam leverages both the first-order moment (the mean of the gradients) and the second-order moment (the variance of the gradients), providing more informed parameter updates. This strategy allows Adam to take into account both the magnitude and the direction of the gradients, improving the stability and efficiency of the learning process [53]. The moments are computed as follows:

$$\begin{aligned} m_k &= \beta_1 m_{k-1} + (1 - \beta_1) \frac{dL}{dw_k}, \\ v_k &= \beta_2 v_{k-1} + (1 - \beta_2) \left(\frac{dL}{dw_k} \right)^2 \end{aligned} \tag{3.14}$$

where m_k and v_k represent the first and second moment estimates, respectively. A bias correction is applied to both estimates to obtain more accurate values, especially during the initial iterations when m_k and v_k are close to zero:

$$\begin{aligned} \hat{m}_k &= \frac{m_k}{1 - \beta_1^k} \\ \hat{v}_k &= \frac{v_k}{1 - \beta_2^k} \end{aligned} \tag{3.15}$$

Finally, the parameter update rule takes the following form:

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{v_k} + \epsilon} m_k \tag{3.16}$$

η is the initial learning rate, which is manually set at the beginning but dynamically adjusted by the Adam optimizer during training. β_1 controls the decay rate of the moving average of the gradients, while β_2 governs the moving average of the squared gradients. Finally, ϵ is a very small constant introduced to ensure numerical stability and to prevent division by zero.
There are numerous advantages of Adam; among them are ease of implementation, computational efficiency, low memory requirements, and high scalability, which make it particularly well-suited for scenarios involving large datasets and high-dimensional models [54].

3.5.5 L-Broyden–Fletcher–Goldfarb–Shanno (L-BFGS)

The L-Broyden–Fletcher–Goldfarb–Shanno (Limited-memory BFGS) method is an optimization algorithm belonging to the family of quasi-Newton methods. It was developed as a variant of the BFGS method to handle large-scale problems with limited memory usage [55]. Unlike Newton's method, which requires the explicit computation of the Hessian matrix, quasi-Newton methods aim to find the minimum of a function by iteratively approximating the Hessian, significantly reducing computational costs. In L-BFGS, the approximation of the Hessian matrix (or more precisely, its inverse) is constructed and updated iteratively by leveraging differences in gradients and positions between successive iterations [56]. This approach is based on a multidimensional generalization of the secant method, which allows estimating the second-order behavior of the objective function without directly computing second derivatives, instead using finite difference approximations. In one dimension, this is expressed by the well-known secant method formula:

$$x_{k+1} = x_k - f'(x_k) \cdot \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})} \tag{3.17}$$

Extending this idea to higher dimensions, the approximated Hessian matrix B_{k+1} satisfies the quasi-Newton condition:

$$B_{k+1}[x_{k+1} - x_k] = \nabla f(x_{k+1}) - \nabla f(x_k), \tag{3.18}$$

Since the secant condition alone does not uniquely determine the update of the inverse Hessian approximation, additional constraints must be imposed [57]. In L-BFGS, these constraints consist of minimizing the change between two successive inverse matrices, that is, minimizing the norm $\|B_{k+1}^{-1} - B_k^{-1}\|$, while ensuring that the updated matrix remains symmetric and positive definite. This strategy ensures numerical stability and adapts to the shape of the objective function, allowing for more effective optimization.

Chapter 4

Physics-Informed Neural Networks

PINNs are a class of deep learning models that incorporate prior physical knowledge into the training process. Unlike traditional neural networks, which rely exclusively on data, PINNs integrate the partial differential equations (PDEs) governing the phenomenon directly into the learning framework [58]. The main objective is to minimize a loss function that combines multiple components: the residuals of the PDE evaluated at selected collocation points within the domain, the residuals associated with the boundary conditions, and, when available, discrepancies from observational data. In this way, the neural network not only learns from data but also respects the physical laws that describe the system. PINNs can, in fact, be employed to address both forward and inverse problems. In forward problems, the network approximates the solution of the governing PDEs using the known physical laws and boundary conditions. In inverse problems, the network is trained on sparse observational data, where the solution is known only at specific locations, and it leverages the underlying physics to infer unknown parameters or uncover hidden system dynamics [59].
A typical PINN architecture consists of an input layer, a variable number of hidden layers with nonlinear activation functions, and an output layer representing the predicted solution of the physical system. The training process relies on the iterative update of the weights connecting the various layers, based on the chosen activation function, with the objective of minimizing a loss that typically includes multiple components: a physics-based term enforcing the governing PDEs, a term related to boundary and initial conditions, and a data-driven term incorporating observational data [22]. The network is trained until the loss function reaches a minimum, or until a predefined maximum number of iterations is reached. This optimization process is typically carried out using algorithms such as ADAM and L-BFGS, which facilitate both rapid convergence and fine-tuning [60].

4.1 The building blocks of PINNs

In this section, the fundamental components of a PINN are presented, revisiting and expanding upon the key concepts introduced in the previous chapter.

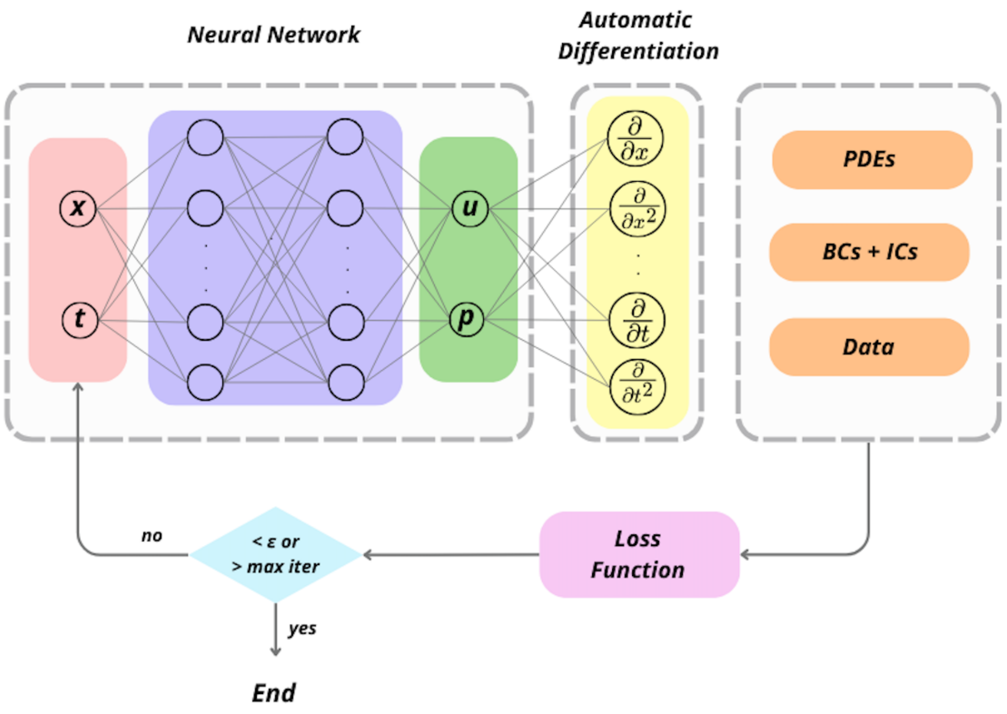

Neural Network
Automatic Differentiation
x
t
u
p
$\frac{\partial}{\partial x}$
$\frac{\partial}{\partial x^2}$
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial t^2}$
PDEs
BCs + ICs
Data
Loss Function
< ε or > max iter
no
yes
End

Figure 4.1: General framework of the operation of a Physics-Informed Neural Network.

4.1.1 Automatic Differentiation

The solution of Partial Differential Equations (PDEs) through PINNs relies on machine learning techniques to efficiently compute derivatives, including gradients and Hessian matrices. Derivative computation methods can be categorized into four main approaches [61]:

1. Manual differentiation, where derivatives are derived analytically and then implemented in code, which is time-consuming and susceptible to human errors.

2. Numerical differentiation, which estimates derivatives using finite difference approximations but suffers from truncation and round-off errors, limiting its accuracy and scalability.

3. Symbolic differentiation, which employs algebraic manipulation in computer algebra systems, yet often leads to expression swell and inefficiency in complex functions.

4. Automatic differentiation (AD), also known as algorithmic differentiation, which efficiently computes derivatives by systematically applying the chain rule at the computational graph level, combining accuracy with computational efficiency.

AD is a powerful technique for efficiently computing derivatives of functions in machine learning [62]. AD leverages the fact that differentiable functions are composed of elementary operations with known derivatives. By systematically applying the chain rule, AD accumulates these derivatives to compute gradients with high precision [63]. AD operates in two primary modes: Forward Mode and Reverse Mode. Forward Mode efficiently computes derivatives for functions with a small number of input variables, while Reverse Mode, widely used in deep learning, is significantly less costly to evaluate when dealing with functions that have many inputs but a single scalar output.

Forward Mode

In Forward Mode Automatic Differentiation, each intermediate variable v_i during function evaluation is augmented with its derivative [64]. Each input variable, known as *primal*, is paired with its corresponding derivative, called *tangent*, denoted by $\dot{v}_i$.

$$v_i \to (v_i, \dot{v}_i)$$

To efficiently compute derivatives, the process begins by initializing the derivative of interest to 1, while all other derivatives are set to 0. As the function is evaluated step by step, the derivatives are propagated using the chain rule, ensuring that both the function values and their corresponding derivatives are updated simultaneously. By the end of the computation, this approach provides both the function's output and its derivative with respect to the chosen input [61].

This mechanism extends to functions with multiple inputs and outputs. For a general function $f : \mathbb{R}^n \to \mathbb{R}^m$, the goal is to compute the Jacobian matrix. To compute the full Jacobian using Forward Mode, n forward passes are required, one for each input direction, resulting in one column of the Jacobian per pass.

A key advantage of Forward Mode is the efficient computation of the Jacobian–vector product $J_f \cdot r$, where $r \in \mathbb{R}^n$, by initializing the tangents as $\dot{x} = r$:

$$J_f r = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix}$$

This allows computing the product in a single pass without explicitly forming the full Jacobian matrix, significantly reducing memory and computational costs.

Reverse Mode

Reverse Mode Automatic Differentiation is a technique that generalizes the backpropagation method, used during weight updates to compute gradients starting from the outputs, with the goal of minimizing the error function. Unlike Forward Mode, which starts from the inputs, Reverse Mode is particularly efficient for functions with many inputs and a single output [65].

In this method, a forward pass is first performed to compute intermediate values and record the dependencies necessary for derivative computation. Then,

a reverse pass is executed to calculate derivatives by applying the chain rule in reverse.
Each intermediate variable v_i is associated with a value called the *adjoint*, denoted $\bar{v}_i$, which represents the partial derivative of the output with respect to v_i. The adjoint of the output is initially set to 1:

$$\bar{y} = \frac{\partial y}{\partial y} = 1,$$

and the computation proceeds by calculating the adjoints of intermediate variables and inputs backward, accumulating contributions when a variable influences the output through multiple paths [66].
For functions with multiple outputs ($f : \mathbb{R}^n \to \mathbb{R}^m$), m reverse passes are required to fully compute the Jacobian. However, similarly to Forward Mode, which computes the Jacobian-vector product, Reverse Mode efficiently computes the vector-Jacobian product, avoiding the explicit computation of the full Jacobian, given by:

$$r^T J_f = [r_1, r_2, \ldots, r_m] \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

4.1.2 Loss Function

In PINNs, the loss function ensures that the model's predictions are consistent with both observed data (when available) and the underlying physical equations. It typically consists of multiple terms, each addressing a different aspect of the problem. These terms can include a term loss that enforces initial and boundary conditions, incorporates data measurements when available, and minimizes the residuals of differential equations, as well as an term loss that includes the physical constraints, specifically the PDEs governing the system, over the space-time domain. The exact formulation of the loss function varies depending on the specific problem, balancing these contributions by adjusting the weights to ensure that the solution is both accurate and physically consistent [67]. Typically, the loss is formulated as a mean squared error function, which must be minimized:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_f} \sum_{i=1}^{N_f} \sum_{j=1}^{4} \left\| \mathbf{f}_j(\mathbf{x}_f^i, t_f^i) \right\|^2, \tag{4.1}$$

where

$$\mathbf{f} = \begin{cases} f_{1,2,3} = \rho \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \mu \Delta \mathbf{u} - \mathbf{f}, \\ f_4 = \nabla \cdot \mathbf{u}. \end{cases}$$

$$\mathcal{L}_{\mathrm{BC}} = \frac{1}{N_{\mathrm{BC}}} \sum_{i=1}^{N_{\mathrm{BC}}} (u(x_i) - \hat{u}_i)^2 \tag{4.2}$$

$$\mathcal{L}_{\mathrm{data}} = \frac{1}{N_{\mathrm{data}}} \sum_{i=1}^{N_{\mathrm{data}}} (u(x_i) - \hat{u}_i)^2 \tag{4.3}$$

- $\mathcal{L}_{\mathrm{PDE}}$ represents the PDE residual loss,
- $\mathcal{L}_{\mathrm{BC}}$ represents the initial/boundary condition loss.
- $\mathcal{L}_{\mathrm{data}}$ represents the data measurements loss.

Here x_i represents the spatial and/or temporal coordinates of the sampled points, $\hat{u}_i$ represents the reference values, and $u(x_i)$ is the predicted solution at those sampled points. The weights w_{PDE}, w_{BC}, and w_{data} are used to balance the contributions of the data loss, boundary conditions, and PDE residual terms within the total loss function, which is minimized by iteratively updating the network's weights through a gradient-based optimization method [22].

Chapter 5

Mathematical Models

This chapter introduces the mathematical models used to describe blood flow dynamics. In particular, it presents the Navier–Stokes equations that govern fluid motion, along with the boundary conditions that characterize the physical domain.

5.1 Navier–Stokes Equations

The Navier–Stokes equations describe the motion of a fluid with constant kinematic viscosity ν. They can be expressed as [68]:

$$\begin{cases} -\nabla \cdot \left[\nu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right] + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{f}, \\ \nabla \cdot \mathbf{u} = 0, \end{cases} \tag{5.1}$$

where:

- $\mathbf{u}$ is the velocity field,
- p is the pressure,
- $\nu = \dfrac{\mu}{\rho}$ is the kinematic viscosity,
- ρ is the fluid density,
- μ is the dynamic viscosity,
- $\mathbf{f}$ is an external force term.

The term $(\mathbf{u} \cdot \nabla)\mathbf{u}$ represents the convective transport of momentum, while the diffusive term $-\nabla \cdot \left[\nu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right]$ represents viscous effects due to molecular diffusion of momentum.
The first equation represents the conservation of momentum, while the second corresponds to the conservation of mass, commonly referred to as the continuity equation.

For unsteady (time-dependent) flows, the Navier–Stokes equations are expressed in the form:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} - \nabla \cdot \left[\nu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right] + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{f}, \\ \nabla \cdot \mathbf{u} = 0, \end{cases} \tag{5.2}$$

If ν is constant and the flow is incompressible (i.e., $\nabla \cdot \mathbf{u} = 0$), then the viscous term simplifies as follows:

$$\nabla \cdot \left[\nu(\nabla \mathbf{u} + \nabla \mathbf{u}^T)\right] = \nu \Delta \mathbf{u}, \tag{5.3}$$

5.1.1 Non-dimensional Navier–Stokes Equations

The Navier–Stokes equations can be expressed in a non-dimensional form, where the dynamics depend on the Reynolds number Re. They are given by:

$$\begin{cases} (\mathbf{u}^* \cdot \nabla)\mathbf{u}^* + \nabla p^* - \dfrac{1}{Re}\Delta \mathbf{u}^* = \mathbf{f}^*, \\ \nabla \cdot \mathbf{u}^* = 0, \end{cases} \tag{5.4}$$

where the non-dimensional variables are defined as:

$$u^* = \frac{u}{U}, \quad x^* = \frac{x}{L}, \quad p^* = \frac{p}{\rho U^2}$$

and the Reynolds number is given by:

$$Re = \frac{\rho U L}{\mu}$$

Here, U and L denote the characteristic velocity and length, respectively. Starred variables (*) represent non-dimensional quantities, whereas unstarred ones are dimensional.

5.1.2 First-Order Navier–Stokes Equations

In large vessels, blood can be approximated as a Newtonian fluid. Under this assumption, by explicitly defining the Cauchy stress tensor, the Navier–Stokes equations can be written using the first-order formulation. In this approach, the momentum equation is expressed as:

$$(\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla \cdot \boldsymbol{\sigma} = \mathbf{f},$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor, defined as:

$$\boldsymbol{\sigma} = p\mathbf{I} - \nu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right),$$

where $\mathbf{I}$ is the identity tensor. Moreover, the pressure is related to the stress tensor through the relation:

$$p = -\frac{1}{2}\,\mathrm{tr}(\boldsymbol{\sigma}).$$

5.1.3 Non-Dimensional First-Order Navier-Stokes Equations

By introducing characteristic scales for length L, velocity U, and pressure $p = \rho U^2$, as previously shown in Section 5.1.1, the first-order momentum equation in its non-dimensional form becomes :

$$(\mathbf{u}^* \cdot \nabla^*)\mathbf{u}^* + \nabla^* \cdot \boldsymbol{\sigma}^* = \mathbf{f}^*,$$

where the non-dimensional Cauchy stress tensor is defined as:

$$\boldsymbol{\sigma}^* = p^*\mathbf{I} - \frac{1}{Re}\left(\nabla^*\mathbf{u}^* + (\nabla^*\mathbf{u}^*)^T\right).$$

Several formulations of the Navier–Stokes equations have been explored. Within the context of PINNs, the advantages offered by each formulation are leveraged. Indeed, selecting the most appropriate formulation is crucial to enhance both convergence speed and the accuracy of the results.
In the problems addressed in the following chapters, the dimensional form of the Navier–Stokes equations, as given in equation 5.1, will be primarily used.
However, reformulating the Navier–Stokes equations in a non-dimensional form, as given in equation 5.1.1, offers significant advantages. Physical variables often span different scales, which can lead to imbalanced loss functions in PINNs, where certain terms dominate due to their relative magnitude. Non-dimensionalization addresses this issue by rescaling all terms to comparable values. The normalized formulation enhances both the numerical stability and convergence behavior of PINNs, making it a valuable strategy when dealing with multi-scale problems or variables with significantly different physical units.
Finally, Least-Squares Finite Element Methods (LSFEM) are based on the minimization of convex functionals constructed from the residuals of the governing equations. In this framework, the PDEs are reformulated as first-order systems [69]. This approach is leveraged in PINNs, where the classical formulation of the equations involves higher-order partial derivatives (e.g., the Laplacian $\nabla^2\mathbf{u}$) introducing significant computational complexity and challenges in optimizing the loss function during training. For this reason, from the perspective of PINNs, expressing the Navier–Stokes equations in their first-order form, as given in equation 5.1.2, can offer advantages in terms of training efficiency and reduced computational cost [70].

5.2 Boundary and Initial Conditions

In order for the problem to be well-posed, initial and boundary conditions must be specified. In the two-dimensional case, the Navier–Stokes equations, together

with the specified boundary conditions, define a well-posed problem in the classical sense: solutions exist, are unique, and maintain continuous derivatives over time without developing singularities, provided the initial data are sufficiently regular. In the three-dimensional case, however, the existence and uniqueness of classical solutions are guaranteed only for limited time intervals [68].
Multiple boundary conditions can be imposed. In this work, we consider conditions imposed either on the velocity field $\mathbf{u}$ or on the normal component of the stress vector $p\mathbf{n} - \nu(\nabla\mathbf{u} + \nabla\mathbf{u}^T)\mathbf{n}$.

5.2.1 Initial Conditions

The initial condition imposed on the domain Ω is typically given by:

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega,$$

where $\mathbf{u}_0$ is a prescribed initial velocity field.

5.2.2 Dirichlet Conditions

In the context of differential problems, Dirichlet boundary conditions can be enforced either strongly, by directly modifying the functional space, or weakly, by incorporating them into the variational formulation.

Strong Form for Boundary Conditions

In differential problems, Dirichlet boundary conditions can be imposed strongly, meaning the numerical solution exactly satisfies the prescribed values on the boundary. Classical numerical methods such as the FEM, FDM, and FVM typically achieve this either by constructing piecewise-defined approximation spaces that inherently satisfy the boundary conditions or by applying a lifting procedure to enforce them.
In the context of PINNs, a similar approach is employed. Since accurately satisfying Dirichlet boundary conditions near the edges of the domain is often challenging and can degrade the solution quality [71], to overcome these issues, hard enforcement it is frequently preferred. This is achieved by constructing the neural network such that it inherently satisfies the given initial and boundary conditions through the addition of a particular solution [72].
The approach consists of designing an auxiliary function that is identically zero on the boundary of the domain, while remaining strictly positive inside the domain. We define the solution as:

$$\hat{u}(x) = g(x) + \ell(x)\, N(x; \theta)\,,$$

where $\hat{u}(x)$ is the network's prediction, $g(x)$ encodes the exact wall conditions, $\ell(x)$ vanishes on the boundary and is positive inside the domain, and $N(x; \theta)$

is the unconstrained network output. This construction enforces the boundary conditions exactly while leaving the network free to learn within the interior of the domain [73].

Weak Form for Boundary Conditions

In FEM, the weak imposition of Dirichlet boundary conditions is achieved by modifying the variational formulation with additional penalty or stabilization terms.
Weak imposition is more natural in PINNs, meaning that instead of constructing a formulation that exactly satisfies the boundary conditions, the optimizer searches for a neural network that minimizes the residual associated with the Dirichlet boundary conditions.

Although the hard way of boundary conditions generally improves the accuracy of the solution, it typically requires explicit knowledge of the domain geometry, often in the form of a mesh or an analytical description of the boundary. On the other hand, the weak enforcement commonly adopted in PINNs may lead to reduced accuracy, as boundary conditions are only approximately satisfied rather than enforced exactly. However, this soft formulation offers significant flexibility, eliminating the need for a mesh or precise geometric information, which represents a clear advantage in problems involving complex or irregular domains.

5.2.3 Pressure Boundary Conditions

In general, Navier-Stokes problems do not prescribe conditions on the pressure. Beyond conditions on the velocity, a set of boundary conditions that can be prescribed are in the form:

$$p\mathbf{n} - \mu\left(\nabla\mathbf{u} + \nabla^T\mathbf{u}\right)\cdot\mathbf{n} = \text{data}$$

where data is a given vector specifying the normal component of the stress tensor. This condition is called "natural" as it is automatically prescribed in the variational form. When data = 0, we say that we prescribe a traction-free or a “do-nothing” condition [74]. When one wants to prescribe a condition on the pressure, the traditional approach is to prescribe the natural condition with:

$$\text{data} = P\mathbf{n}$$

where P is the pressure data to prescribe.
In the context of PINN, where we rely on the least square minimization of the residulas, we can actually prescribe directly the pressure adding a term on the loss function

$$w_{PBC}\|p - P\|^2_{\Gamma_{PBC}}$$

where Γ_{PBC} is the portion of the boundary where we want to prescribe the pressure. We will explore this option in the subsequent chapters.

Chapter 6

The Clinical Problem: The Relevance of Wall Shear Stress

Blood flow within blood vessels exerts mechanical forces on the vascular wall, which can be primarily divided into two components: the perpendicular component, mainly due to blood pressure, which can cause structural deformations of the vessel wall cells, and the tangential component, known as *Wall Shear Stress*, which represents the shear force exerted parallel to the endothelial surface [75]. WSS is sensed by the endothelium through mechanotransduction mechanisms, which convert mechanical stimuli into biochemical signals that regulate the structure and function of the vascular wall [2, 76]. Under physiological conditions, WSS plays a crucial role in maintaining vascular homeostasis, regulating vessel diameter and inhibiting pathological processes such as cellular proliferation, thrombosis, and inflammation of the vascular wall [77]. However, in the presence of disturbed flow, characterized by changes in the direction of flow over time and space, WSS becomes abnormal, assuming oscillatory or reduced values [78]. These alterations impair endothelial function and are closely linked to the onset and progression of cardiovascular diseases, particularly atherosclerosis. Given the strong correlation between WSS and both the presence and development of vascular pathologies, evaluating this parameter is of fundamental importance. This is made possible through the analysis of vessel morphology using advanced imaging techniques, such as X-ray angiography or computed tomography angiography (CTA), combined with high-resolution intravascular imaging, such as optical coherence tomography (IV-OCT) or intravascular ultrasound (IVUS) [79].

6.1 Computation of Wall Shear Stress

The standard method for modeling blood flow is patient-specific CFD, which uses data acquired from angiography and other imaging modalities. By numerically solving the Navier-Stokes equations, CFD computes velocity and pressure fields at each point in space and time. This is particularly critical for estimating WSS, a quantity that cannot be measured directly as it depends on the spatial gradient of velocity near the vessel wall. In general, the instantaneous WSS can be expressed as follows [80]:

$$\boldsymbol{\tau}_w(\mathbf{x},t) \equiv \mu \left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right) \cdot \mathbf{n} - \mu \left[\mathbf{n} \cdot \left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right) \cdot \mathbf{n}\right] \cdot \mathbf{n}, \tag{6.1}$$

where μ is the dynamic viscosity, $\mathbf{u}$ is the velocity field, and $\mathbf{n}$ is the outward unit normal to the vessel wall. This expression represents the tangential component of the normal stress vector.
The spatially averaged WSS over a surface of interest Σ is defined as:

$$\overline{\boldsymbol{\tau}}(t) = \frac{1}{|\Sigma|} \int_\Sigma \boldsymbol{\tau}_w(\mathbf{x},t)\, d\mathbf{x}, \tag{6.2}$$

where $|\Sigma|$ denotes the area of the surface Σ.
In the steady case, the WSS is computed directly as a spatial average:

$$WSS = \frac{1}{|\Sigma|} \int_\Sigma \boldsymbol{\tau}_w(\mathbf{x})\, d\mathbf{x}. \tag{6.3}$$

In some cases, a time-averaged value, known as the Time-Averaged Wall Shear Stress (TAWSS), is used. It can be expressed as a spatial average over a surface Σ and a period of time T:

$$TAWSS \equiv \frac{1}{T} \int_0^T \overline{\boldsymbol{\tau}}(t)\, dt, \tag{6.4}$$

where T is the duration of the cardiac cycle.
Although CFD remains the gold standard for simulating vascular flow, it still presents several challenges. First, mesh generation for complex vascular geometries is still not fully automated and often requires manual refinement, especially in intricate anatomical regions. Moreover, CFD results are highly sensitive to the initial conditions and to the input data quality, such as imaging-based vessel segmentation, which is subject to uncertainty. Additionally, the computational cost of advanced CFD simulations is high: even the simplest simulations require several minutes to generate results, while more complex, patient-specific cases with detailed boundary conditions can take considerably longer to compute [78].
To overcome these limitations, deep learning methods, particularly PINNs, have gained increasing attention in recent years. PINNs are capable of predicting velocity and pressure fields that satisfy the governing physical equations without the need for dense data or traditional meshing. This makes them a mesh-free alternative to classical CFD approaches, especially useful in situations where data are sparse, incomplete, or of low resolution. While promising, this technology is still under development and not yet ready for widespread clinical application [16].

6.2 Wall Shear Stress and Clinical Implications

6.2.1 Atherosclerosis

Atherosclerosis is a chronic vascular disease characterized by inflammation of the intima layer of the arteries, leading to the formation of atherosclerotic plaques.

These plaques are composed of lipids, inflammatory cells, and collagen fibers, which accumulate and can cause narrowing of the arterial lumen, reducing blood flow. Atherosclerosis can evolve into serious clinical conditions, such as myocardial infarction, ischemic stroke, aneurysm, and peripheral artery disease [81]. The causes of the disease are multifactorial, with modifiable risk factors such as smoking, obesity, elevated cholesterol levels, and diabetes mellitus, as well as non-modifiable risk factors, including age, sex, and genetics [82]. In addition to these, local biomechanical factors, such as vessel diameter, curvature, and tortuosity, play a crucial role in the onset and progression of the disease [83].

The standard method for treating patients with atherosclerosis is percutaneous coronary intervention (PCI) with the implantation of drug-eluting stents (DES). Despite significant technological advances in recent years, the occurrence of in-stent restenosis (ISR) and stent thrombosis (ST) continues to pose a major clinical challenge. Indeed, the drug-eluting stent interacts with the vessel wall, modulating the local cellular response, which can lead to neointimal proliferation and, in the medium to long term (about a year after implantation), contribute to the development of a new atherosclerotic plaque within the stent, resulting in restenosis of the vascular lumen [84]. The implantation of the stent significantly alters the local hemodynamics, causing disturbances in blood flow, such as recirculation zones and disturbed flow, particularly near the struts, the metal structures that make up the stent. Furthermore, in the case of malapposition, where the stent is not in uniform contact with the vessel wall, an accentuation of turbulence and vortex formation is observed. These areas, characterized by disturbed flow, constitute environments favorable to thrombus formation [85]. It has also been shown that the geometric design of the stent, the materials used, and the implantation technique significantly affect the post-procedural hemodynamic environment. This has led to research focused on the development of next-generation stents designed to minimize flow alterations and reduce the occurrence of post-procedure complications.

Based on these considerations, it is clear that atherosclerosis and the follow-up of the intervention are influenced by hemodynamics. From a clinical perspective, WSS is a particularly relevant hemodynamic metric, as alterations in blood flow, especially in the presence of complex or tortuous vascular geometries, can favor the onset and progression of atherosclerotic disease. Moreover, WSS is a crucial quantity in post-operative follow-up, as it allows for a detailed analysis of the interaction between the stent and local flow patterns. By simulating blood flow dynamics, it is possible to directly assess how the specific geometry and positioning of the stent affect blood flow, as well as to estimate the risk of thrombogenesis and the potential pro-restenotic effects of the implant [86].

6.2.2 Biomechanical Factors

Variations in blood vessel geometry significantly influence blood flow dynamics, leading to the occurrence of disturbed flow patterns characterized by vortices, recirculation zones, and helical motion. These geometric alterations disrupt normal laminar flow, transforming it into chaotic and irregular flow. Such disturbed flow induces WSS that varies in both magnitude and direction along the vessel's inner surface. These fluctuations in WSS affect the behavior of endothelial cells, which line the vessel wall, triggering biological responses that may promote the onset and progression of atherosclerotic plaques. Consequently, regions marked by geometric irregularities, such as stenoses, tortuosity, and bifurcations, are most frequently associated with the development of these lesions [78]. From a clinical perspective, the analysis of vascular geometry is essential for assessing the risk of developing atherosclerotic lesions. Certain geometric parameters, which can be easily detected through standard imaging techniques, may serve as predictive indicators of susceptibility to these pathologies [87].

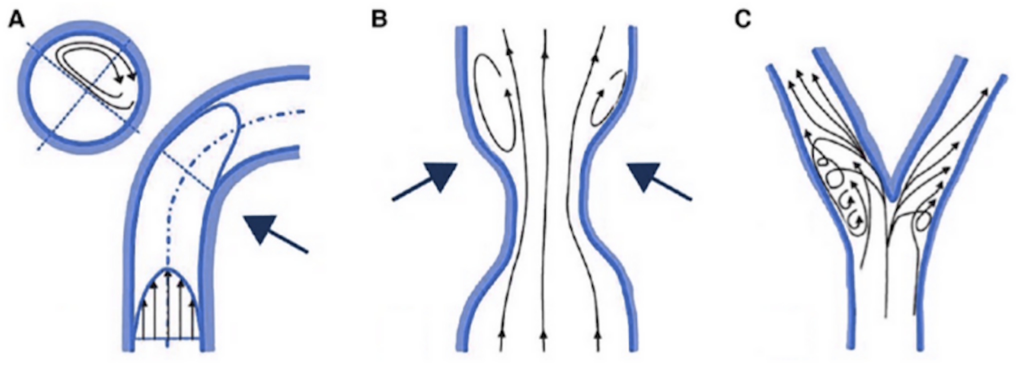


Figure 6.1: The figure illustrates three different vascular conditions that promote disturbed flow and the formation of atherosclerotic plaques. Figure A shows a tortuous vessel, which creates irregular flow; Figure B represents a stenosis, which accelerates the flow and generates turbulence; Figure C illustrates a bifurcation, which alters the direction of flow [83].

Curvature and Tortuosity

The dynamics of blood flow are strongly influenced by the curvature and torsion of blood vessels. Certain geometric indicators, such as the radius of curvature, the distance between successive bends, and the curvature angle, play a crucial role in determining local hemodynamic conditions and assessing the risk of lesion formation [88]. In regions with high local curvature, blood recirculation phenomena are observed, leading to variations in flow velocity. Specifically, higher velocities are recorded along the outer wall of the vessel curvature, while lower velocities are generally found along the inner wall. This also affects pressure losses: in particular, when considering a curved tube, the pressure drop is greater than in a straight tube of equal length and flow rate. This is because torsion breaks the symmetry of the helical flows induced by curvature, causing one of the helical structures to become dominant [89]. This phenomenon is evident in coronary ar-

teries, where increasing vessel tortuosity, a global geometric property describing the overall winding or twisting of the vessel, leads to greater pressure drops and may therefore reduce blood perfusion [90].
However, the role of tortuosity remains complex and not yet fully understood. Some studies have shown that increased tortuosity contributes to the progression of atherosclerosis. Specifically, small radii of curvature, short inter-bend distances, and high curvature angles, all local geometric features, are associated with lower WSS, a factor linked to the development of atherosclerotic disease [91]. Conversely, other studies suggest that spiral (helical) tortuosity may mitigate the risk of atherosclerosis compared to planar tortuosity, by promoting more uniform shear stress distribution [92].

Stenosis

The dynamics of blood flow are significantly altered in the presence of stenosis. The narrowing of the vascular lumen modifies the normal flow profile, causing it to lose its symmetry upon encountering the obstruction. In this context, the phenomenon of convective acceleration occurs, characterized by an increase in flow velocity and a simultaneous decrease in pressure downstream of the stenosis; subsequently, the flow tends to decelerate [78]. According to Bernoulli's principle, the pressure drop observed upstream of the stenosis should, in theory, be completely recovered downstream. However, in the physiological reality, this does not occur, as part of the mechanical energy is dissipated in the form of heat due to energy losses from viscous friction that develops through the stenosis [93]. The inefficiency in energy recovery is also associated with the formation of vortical currents and turbulent structures, manifestations of the breakdown of the laminar flow typically seen in healthy, non-stenotic vessels [94]. These alterations directly impact the WSS, which varies significantly depending on the position relative to the stenosis. In the pre-stenotic phase, WSS is generally low and oscillating due to the presence of recirculation; in the post-stenotic phase, these oscillations become more pronounced. In contrast, at the point of maximum narrowing, the so-called "neck" of the stenosis, a peak maximum of WSS is observed due to the acceleration of blood flow [95]. Finally, it is well established, as previously highlighted, that regions characterized by low and highly oscillatory WSS values represent pro-atherogenic environments, as they promote the onset and progression of atherosclerotic plaques. Consequently, the presence of stenosis not only represents an already advanced pathological condition but also contributes to further deterioration of the situation, triggering mechanisms that may lead to the worsening of the clinical picture.

Bifurcation

Vessel bifurcations are particularly sensitive to the development of pathology, as these regions of the vessel present secondary flow zones and helical flow, which may promote the formation of atherosclerotic lesions. Anatomically, a coronary bifurcation consists of three segments: a proximal main branch (Main Vessel, MV), which divides at a point called the carina into two distal branches, namely the distal main branch (Distal Main, DM) and the lateral branch (Side Branch, SB) [96]. Hemodynamically, the bifurcation causes a deviation in the blood flow from the main vessel, resulting in an asymmetric velocity profile. In particular, higher velocities are observed on the inner side of the lateral branch, continuous with the carina, and lower velocities are seen on the opposite walls of the MV and SB. This condition leads to low and oscillating wall shear stress, which promotes the development of disturbed flow, characterized by recirculations, flow separation, and vortical structures [97]. Geometric factors can also influence flow patterns and WSS distribution. Specifically, the bifurcation angle has a limited impact on hemodynamics [87], whereas a reduced curvature radius is associated with higher intensity of helical flow, accompanied by symmetry loss. This effect increases with distance from the bifurcation region and leads to greater exposure to low WSS, especially in stenotic models [98].

6.2.3 Wall Shear Stress Thresholds

It has been observed that WSS represents a crucial quantity, the value of which can serve as an early indicator of physiological or potentially pathological conditions of blood flow. In particular, low WSS values have been correlated with the initiation, progression, and instability of atherosclerotic plaques, promoting an inflammatory and atherogenic microenvironment. On the other hand, high WSS values are generally considered atheroprotective, as they favor a stable endothelial phenotype; however, under certain circumstances, they may contribute to plaque rupture [99]. For the clinical classification of WSS, a commonly accepted threshold system has been proposed: values lower than 1 Pa are considered indicative of low WSS, values between 1 and 7 Pa are categorized as normal-high, and values greater than 7 Pa are classified as high WSS [100]. It is important to note that alternative threshold values have been proposed in the literature, often varying depending on the specific vascular region or measurement technique [85].

Chapter 7

Application of PINNs on 2D Geometries

In this section, we test the effectiveness of PINNs in solving fluid flow dynamics, emphasizing the role of data assimilation in enhancing accuracy. Specifically, we apply PINNs to two key problems: *Womersley flow*, which is crucial for studying pulsatile blood flow in arteries such as the aorta and carotid arteries [101], and the *Kim&Moin* problem, a standard case used to assess methods for solving the incompressible Navier–Stokes equations [102].
To thoroughly assess the performance of PINNs, we explore various parameters to fine-tune each configuration according to the specific characteristics of the problem under investigation. In addition, we investigate the impact of data assimilation by incorporating velocity and/or pressure observations into the training process, enabling the model to reconstruct missing information from sparse, incomplete, or noisy measurements. Moreover, we tested a configuration without boundary conditions, using only observational data and the governing equations. This choice was motivated by the fact that, in hemodynamics, boundary conditions are often unknown. Therefore, we wanted to investigate whether PINNs can still perform well in their absence.
To ensure a comprehensive evaluation, we perform experiments under five distinct scenarios:

1. Without data: the PINN model is trained purely based on the governing equations, without any additional measurement data.

2. With velocity data: a set of noise-free velocity data is incorporated in the model.

3. With velocity and pressure data: a set of noise-free velocity and pressure data is incorporated in the model.

4. With noisy data: realistic conditions are simulated by adding white noise with different standard deviations to the exact solution, evaluating the robustness of the approach under varying levels of noise.

5. With data but without boundary conditions: a set of noise-free velocity and pressure data is incorporated into the model, even in the absence of explicitly defined boundary conditions.

7.1 Methods

7.1.1 Network Configuration

For both problems considered, a feed-forward neural network was employed using the DeepXDE library [103]. The architecture consists of four hidden layers with 100 neurons each, utilizing the *tanh* activation function. Optimization was carried out using a combination of the *Adam* and *L-BFGS* algorithms, with a learning rate of 1×10^{-4}. The network takes three input variables (x, y, t) and outputs three quantities (u, v, p), corresponding to the two velocity components and the pressure. For simplicity, the same network was adopted to predict both velocity and pressure fields.

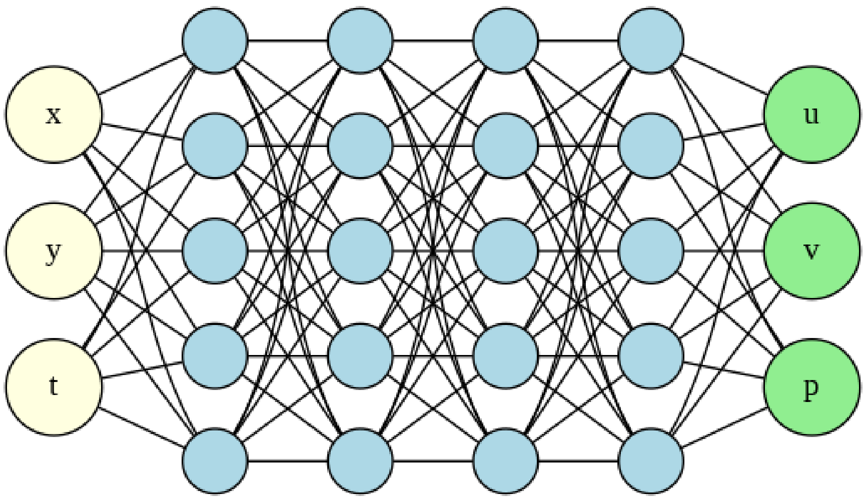


Figure 7.1: Architecture of the Physics-Informed Neural Network. The network consists of three input layers (x, y, t), representing the spatial and temporal coordinates, followed by four hidden layers with a fixed number of neurons. Finally, the output layers (u, v, p), corresponding to the velocity components and pressure.

7.1.2 Evaluation of Different Training Configurations for PINNs

Beyond the fixed network configuration presented, we systematically explore the impact of several key hyperparameters on the network's performance. Specifically, we analyze the influence of the number of collocation points within the

domain and on the boundaries, the number of training iterations in the loss function to assess their effect on the ultimate accuracy and convergence of the model. Moreover, we assess the distribution of the collocation points, which can be either randomly sampled or uniformly distributed across the spatial domain. For unsteady problems, these points are spread over the entire temporal interval, ensuring that the network learns the full time evolution of the solution. To enhance the model's generalization capabilities and avoid overfitting to the initially selected collocation points, a resampling strategy is implemented every 1000 iterations. By periodically updating the training points, this approach prevents the network from becoming too specialized on a fixed set of locations, thus mitigating the risk of overfitting. Instead, it encourages exploration across different regions of the domain, enabling the network to better capture the overall behavior of the solution and improve its accuracy.
For data assimilation, we examine the effect of the number of observational data points and evaluate whether incorporating only velocity data or both velocity and pressure data improves the model's accuracy. The number of observation points was chosen empirically. Additionally, data assimilation was performed using noisy observations by adding Gaussian noise with a reasonable standard deviation to the exact solution of the problem.

Training without data

This table presents different configurations where the model is trained using only the governing equations and boundary conditions. We analyze the effect of varying the number and distribution of collocation points and iterations on the model's performance.

Table 7.1: Configurations with Only PDEs and Boundary Conditions

Configuration	**Collocation Points**	**Iterations**
Random Distribution	20000 (Domain), 2500 (Boundary)	40000
Uniform Distribution	20000 (Domain), 2500 (Boundary)	40000
Increased Collocation Points	40000 (Domain), 5000 (Boundary)	40000
Fewer Collocation Points	10000 (Domain), 1250 (Boundary)	40000
More Training Iterations	20000 (Domain), 2500 (Boundary)	60000
Fewer Training Iterations	20000 (Domain), 2500 (Boundary)	20000

Training with velocity data

The configurations below incorporate observational velocity data into the training process in addition to the PDEs and boundary conditions. We analyze the network's ability to handle velocity-only data and the effect of varying the number of collocation points.

Table 7.2: Configurations with Data Assimilation (velocity data)

Configuration	Collocation Points	Data Points
Velocity Data	20000 (Domain), 2500 (Boundary)	5000
Velocity Data	20000 (Domain), 2500 (Boundary)	10000

Training with velocity and pressure data

The configurations below incorporate observational velocity and pressure data into the training process in addition to PDEs and boundary conditions. We analyze the effect of varying the number of collocation points.

Table 7.3: Configurations with Data Assimilation (velocity and pressure data)

Configuration	Collocation Points	Data Points
Velocity/Pressure Data	20000 (Domain), 2500 (Boundary)	5000
Velocity/Pressure Data	20000 (Domain), 12500 (Boundary)	10000

Training with noisy data

Here, observational data is incorporated alongside PDEs and boundary conditions, with noise introduced to assess the model's robustness. Different noise levels are applied to the velocity and pressure data by varying the standard deviation.

Table 7.4: Configurations with Noisy Data Assimilation

Configuration	Collocation Points	Data Points
Velocity Data (std = 0.1)	20000 (Domain), 2500 (Boundary)	5000
Velocity Data (std = 0.2)	20000 (Domain), 2500 (Boundary)	5000
Velocity/Pressure Data (std=0.1)	20000 (Domain), 2500 (Boundary)	5000
Velocity/Pressure Data (std=0.2)	20000 (Domain), 2500 (Boundary)	5000

Training with data without boundary conditions

Here, the PINN model is trained solely on the governing equations and observational data, without explicitly enforcing boundary conditions. This approach challenges the network to infer the underlying flow dynamics purely from available measurements.

Table 7.5: Configurations with Data Assimilation without Boundary Condiotions

Configuration	Collocation Points	Data Points
Velocity Data	20000 (Domain), 2500 (Boundary)	5000
Velocity/Pressure Data	20000 (Domain), 2500 (Boundary)	5000

7.2 Kim&Moin Problem

7.2.1 Governing Equations

The flow is governed by the unsteady-state Navier-Stokes equations for an incompressible fluid 5.2.
The dynamic viscosity and the density of the fluid are set to 0.01 and 1, respectively [104].

7.2.2 Computational Domain

For the numerical experiments, we define a 2D domain as follows:

- The spatial domain is a square region given by $[0.25, 1.25] \times [0.5, 1.5]$,
- The time domain is considered within the specified range $[0, 2]$, where 2 is the total simulation time.

7.2.3 Definition of Boundary Conditions

In the considered model, specific boundary conditions are imposed to define the behavior of the flow within the two-dimensional domain.

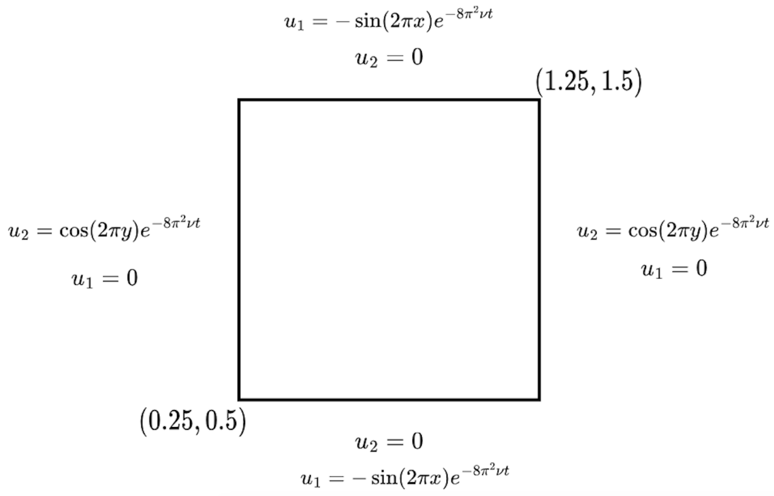

$u_1 = -\sin(2\pi x)e^{-8\pi^2\nu t}$
$u_2 = 0$
$(1.25, 1.5)$
$u_2 = \cos(2\pi y)e^{-8\pi^2\nu t}$
$u_1 = 0$
$u_2 = \cos(2\pi y)e^{-8\pi^2\nu t}$
$u_1 = 0$
$(0.25, 0.5)$
$u_2 = 0$
$u_1 = -\sin(2\pi x)e^{-8\pi^2\nu t}$

Figure 7.2: Representation of the computational domain used in the Kim&Moin simulation. The domain is a two-dimensional square $[0.25, 1.25] \times [0.5, 1.5]$, where boundary conditions are applied to simulate flow.

- Inlet Boundary Conditions ($x = 0.25$):

$$u_1 = 0, \quad u_2 = \cos(2\pi y)e^{-8\pi^2\nu t}$$

- Outlet Boundary Conditions ($x = 1.25$):

$$u_1 = 0, \quad u_2 = \cos(2\pi y)e^{-8\pi^2 \nu t}$$

- Wall Boundary Conditions (lower $y = 0.5$ and upper $y = 1.5$):

$$u_2 = 0, \quad u_1 = -\sin(2\pi x)e^{-8\pi^2 \nu t}$$

7.2.4 Data Assimilation

The data were derived from the exact solutions of the velocity and pressure fields, modeled using the following analytical expressions for u, v, and p [104]:

$$u_1 = -\cos(2\pi x)\sin(2\pi y)e^{-8\pi^2 \nu t} \tag{7.1}$$

$$u_2 = \sin(2\pi x)\cos(2\pi y)e^{-8\pi^2 \nu t} \tag{7.2}$$

$$p = -\frac{1}{4}\left(\cos(4\pi x) + \cos(4\pi y)\right)e^{-16\pi^2 \nu t} \tag{7.3}$$

7.3 Womersley Problem

The Womersley function describes the transient, pulsatile velocity profile of blood flow in circular ducts. It is an exact solution to the viscous flow equations under a periodic pressure gradient and has been widely applied in hemodynamics, particularly in computational models of arterial and vascular blood flow [101].
To test the network, a 2D version of the 3D Womersley model was adopted by formulating the problem in two dimensions, using a square domain to represent the 2D geometry.

7.3.1 Governing Equations

The flow is governed by the unsteady-state Navier-Stokes equations for an incompressible fluid 5.2.
The fluid's dynamic viscosity and density are set to 0.1 and 1, respectively [101].

7.3.2 Computational Domain

For the numerical experiments, we define a 2D domain as follows:

- The spatial domain is a square region given by $[0, 1] \times [0, 1]$,
- The time domain is considered within the specified range $[0, 2]$, where 2 is the total simulation time.

7.3.3 Definition of Boundary and Initial Conditions

In the considered model, specific boundary and initial conditions are imposed to define the behavior of the flow within the two-dimensional domain $[0,1] \times [0,1]$.

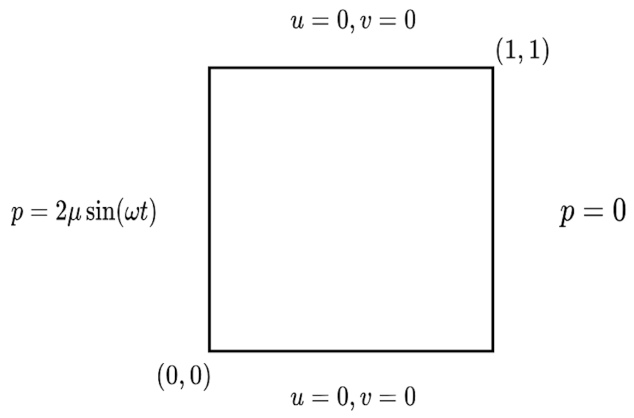


Figure 7.3: Representation of the computational domain used in the Womersley simulation. The domain is a two-dimensional square $[0,1] \times [0,1]$, where boundary conditions are applied to simulate pulsatile flow.

- Inlet Boundary Condition:
  For the pressure, boundary conditions are imposed on the horizontal boundaries of the domain. At the domain inlet ($x = 0$), the pressure is prescribed with a time-dependent oscillatory profile to model a varying pressure gradient characteristic of pulsatile flow:
  $$p = 2\mu \sin(\omega t)$$
- Outlet Boundary Condition:
  At the domain outlet ($x = 1$), the pressure is set to zero:
  $$p = 0$$
- Wall Boundary Condition:

  No-slip conditions are applied on the domain walls ($y = 0$ and $y = 1$) for the velocity components:
  $$u = 0, \quad v = 0$$
  simulating solid walls that prevent fluid motion at these boundaries.
- Initial Condition:
  To ensure proper initialization, the fluid is assumed to be at rest initially throughout the domain:
  $$p(x, y, 0) = 0$$

7.3.4 Data Assimilation

The data were derived from the exact solutions of the velocity and pressure fields, modeled using the following analytical expressions for u, v, and p:

$$
\begin{aligned}
u = \sum_{n=0}^{k-1} & \frac{8\mu \left[\omega \exp\left(-\mu(2n+1)^2\pi^2 t\right) - \omega \cos(\omega t)\right.}{\mu^2(2n+1)^4\pi^4 + \omega^2} \\
& \left. + \mu(2n+1)^2\pi^2 \sin(\omega t)\right] (2n+1)\pi \sin((2n+1)\pi y)
\end{aligned} \tag{7.4}
$$

$$v = 0 \tag{7.5}$$

$$p = 2\mu \sin(\omega t)(1-x) \tag{7.6}$$

The velocity solution is approximated by truncating the series at a finite number of terms k, although the exact solution corresponds to the limit as $k \to \infty$.

7.3.5 Error Analysis

The relative errors are computed according to the formula:

$$\text{Relative } l_2 \text{ Error} = \frac{\|y_{\text{true}} - y_{\text{pred}}\|_2}{\|y_{\text{true}}\|_2} \tag{7.7}$$

Absolute errors are computed as follows:

$$\text{Absolute Error} = |y_{\text{true}} - y_{\text{pred}}| \tag{7.8}$$

where y_{true} represents the reference solution, and y_{pred} is the predicted one.

7.4 Results

7.4.1 Training Time

The average training time is approximately one hour for the cases considered. The number of collocation points has the strongest impact on training time, followed by the number of training iterations, with an increase of around 30 minutes. An increase in either of these parameters leads to a significant rise in computational cost.

The following subsections report the results obtained from the five training configurations described in Section 7.1.2. The accuracy of each model configuration is quantified through the relative l_2 error for the velocity components (e_{u_1}, e_{u_2}) and the pressure field (e_p).

7.4.2 Error Analysis for Kim&Moin Problem

Table 7.6: Relative l_2 Errors – Only PDEs and Boundary Conditions

Configuration	e_{u1} (Velocity x)	e_{u2} (Velocity y)	e_p (Pressure)
Random Distribution	0.1065	0.5712	2.8088
Uniform Distribution	0.1187	0.6232	3.3958
Increased Collocation Points	0.1169	0.2807	1.3099
Fewer Collocation Points	0.3951	0.7447	1.2760
More Training Iterations	0.1179	0.2227	0.9253
Fewer Training Iterations	0.2815	0.5371	0.8481

Table 7.7: Relative l_2 Errors – Data Assimilation (velocity data)

Configuration	e_{u1} (Velocity x)	e_{u2} (Velocity y)	e_p (Pressure)
Velocity Data (5000 pts)	0.0137	0.0190	0.9465
Velocity Data (10000 pts)	0.0116	0.0168	1.6081

Table 7.8: Relative l_2 Errors – Data Assimilation (velocity and pressure data)

Configuration	e_{u1} (Velocity x)	e_{u2} (Velocity y)	e_p (Pressure)
Velocity/Pressure Data (5000 pts)	0.0096	0.0127	0.0222
Velocity/Pressure Data (10000 pts)	0.0098	0.0117	0.0232

Table 7.9: Relative l_2 Errors – Noisy Data Assimilation

Configuration	e_{u1} (Velocity x)	e_{u2} (Velocity y)	e_p (Pressure)
Velocity Data (std = 0.1)	0.0113	0.0132	0.6917
Velocity Data (std = 0.2)	0.0118	0.0153	2.5429
Velocity/Pressure Data (std = 0.1)	0.0178	0.0150	0.0258
Velocity/Pressure Data (std = 0.2)	0.0104	0.0134	0.0258

Table 7.10: Relative l_2 Errors – Data Assimilation without Boundary Conditions

Configuration	e_{u1} (Velocity x)	e_{u2} (Velocity y)	e_p (Pressure)
Velocity Data (5000 pts)	0.0227	0.0239	2.5581
Velocity/Pressure Data (5000 pts)	0.0226	0.0229	0.0456

7.4.3 Error Analysis for Womersley Flow

Table 7.11: Relative l_2 Errors – Only PDEs and Boundary Conditions

Configuration	e_u (Velocity x)	e_p (Pressure)
Random Distribution	0.8788	0.0263
Uniform Distribution	0.7709	0.0252
Increased Collocation Points	0.2890	0.0248
Fewer Collocation Points	0.7016	0.0234
More Training Iterations	0.3339	0.0208
Fewer Training Iterations	0.2641	0.0523

Table 7.12: Relative $_2$ Errors – Data Assimilation (velocity data)

Configuration	e_u (Velocity x)	e_p (Pressure)
Velocity Data (5000 pts)	0.0187	0.0270
Velocity Data (10000 pts)	0.0198	0.0263

Table 7.13: Relative l_2 Errors – Data Assimilation (velocity and pressure data)

Configuration	e_u (Velocity x)	e_p (Pressure)
Velocity/Pressure Data (5000 pts)	0.0191	0.0181
Velocity/Pressure Data (10000 pts)	0.0219	0.0187

Table 7.14: Relative l_2 Errors – Noisy Data Assimilation

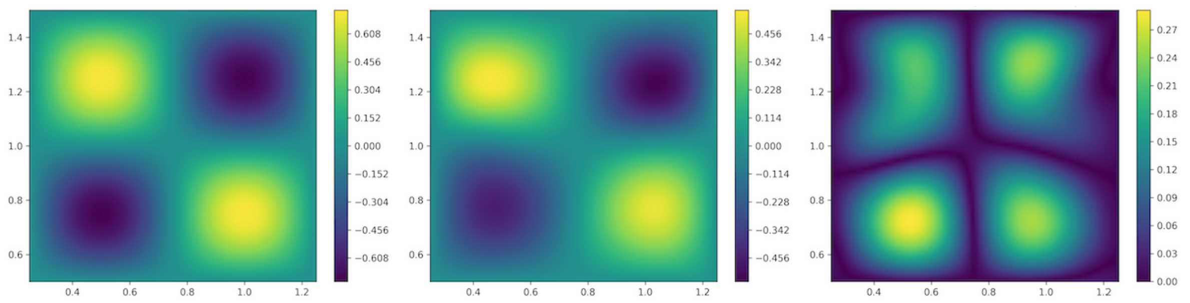
Configuration	e_u (Velocity x)	e_p (Pressure)
Velocity Data (std = 0.1)	0.0169	0.0426
Velocity Data (std = 0.2)	0.0176	0.0181
Velocity/Pressure Data (std = 0.1)	0.0190	0.0195
Velocity/Pressure Data (std = 0.2)	0.0218	0.0175

Table 7.15: Relative l_2 Errors – Data Assimilation without Boundary Conditions

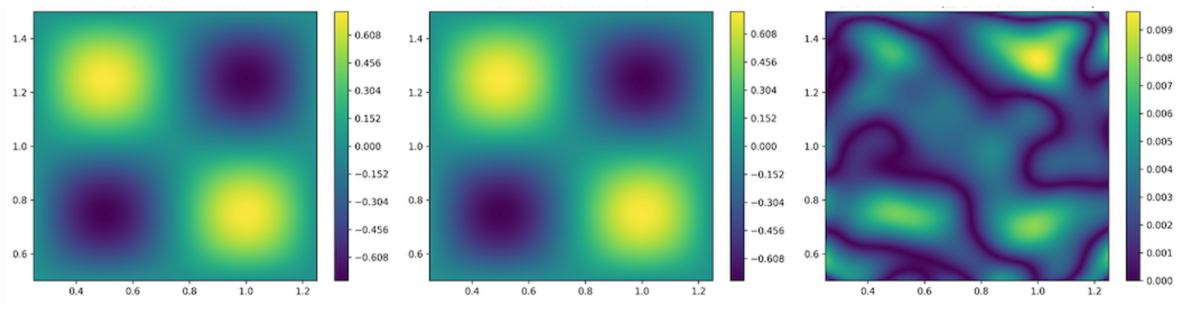
Configuration	e_u (Velocity x)	e_p (Pressure)
Velocity Data (5000 pts)	0.2301	0.9723
Velocity/Pressure Data (5000 pts)	0.0922	0.0418

7.4.4 Kim&Moin Problem

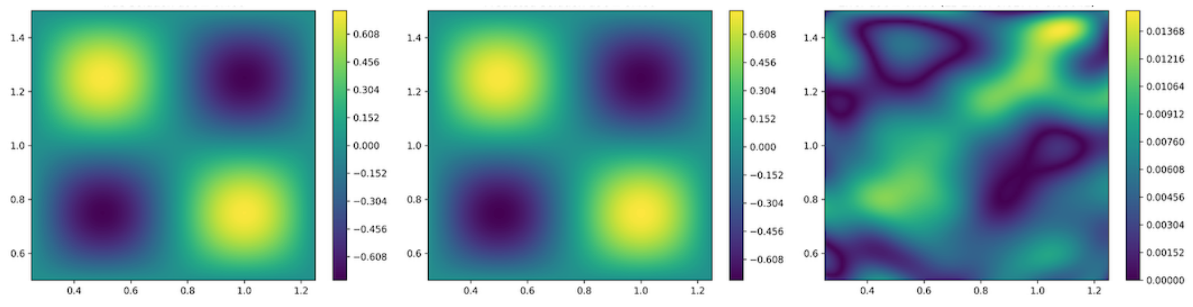
Velocity u_1



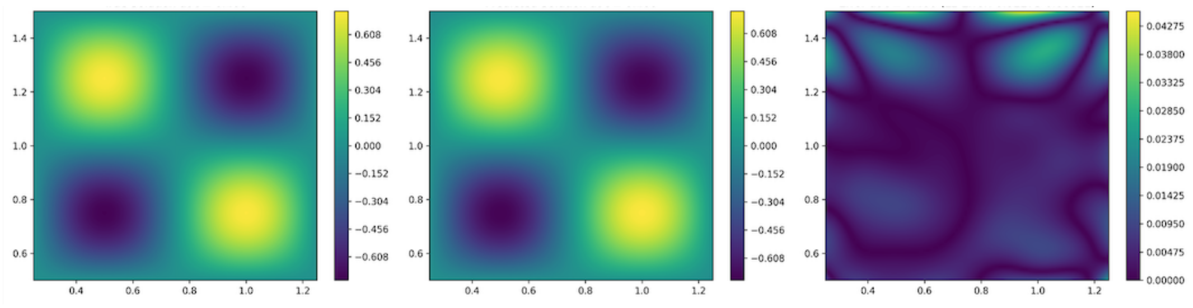
(a) Case A: without data



(b) Case B: with velocity and pressure data



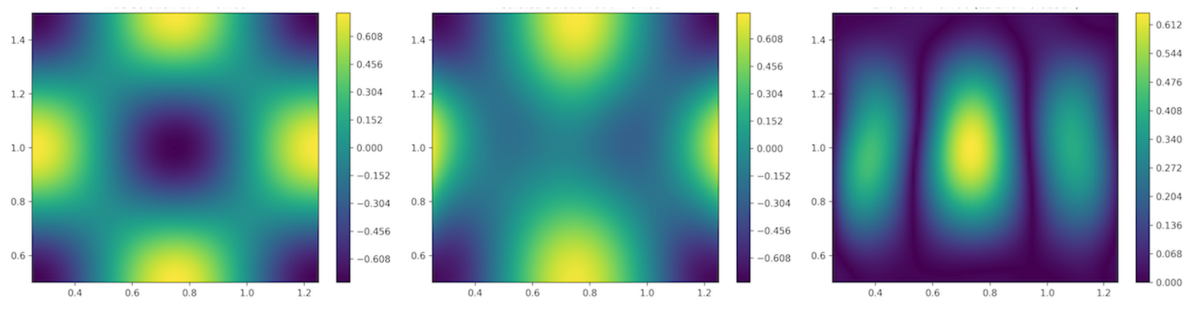
(c) Case C: with noisy velocity and pressure data



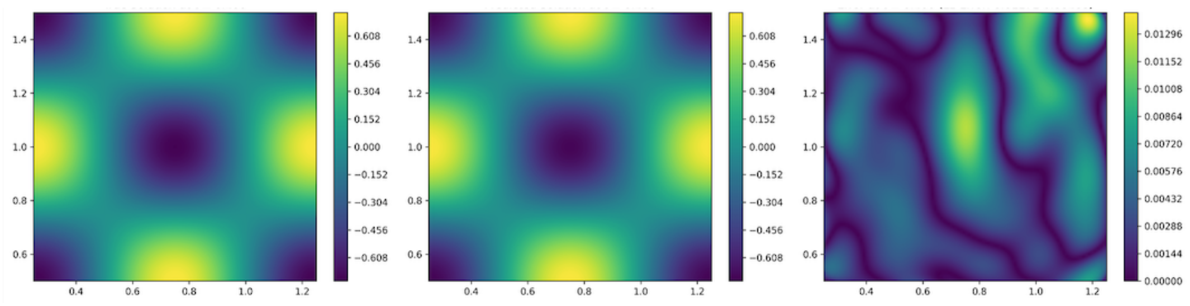
(d) Case D: without boundary conditions

Figure 7.4: Comparison of the velocity u_1 fields for different test cases (a–d) at the same time instant. For each case, the subpanels show, in order: the ground truth solution, the neural network predicted solution, and the absolute error between the two. Note that the scales differ between the panels.

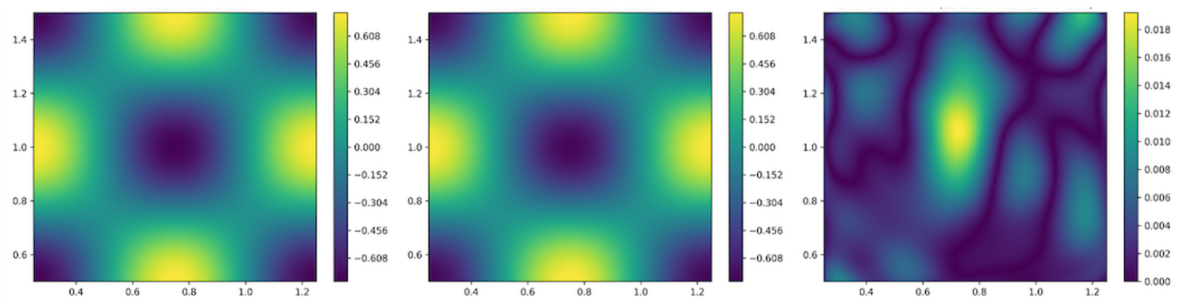
Velocity u_2



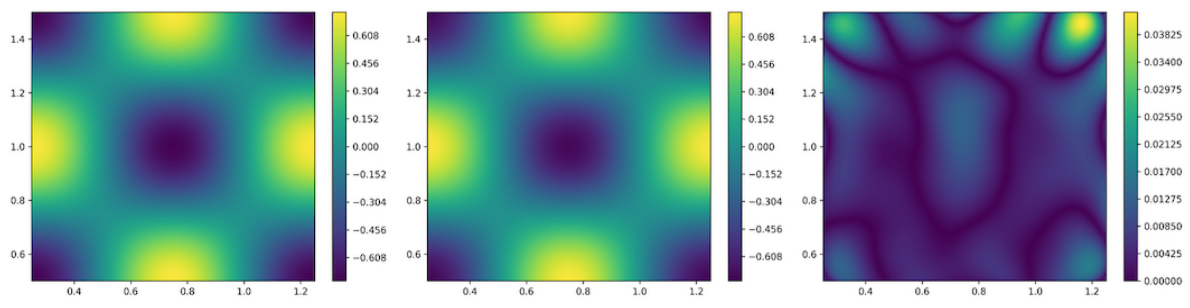
(a) Case A: without data



(b) Case B: with velocity and pressure data



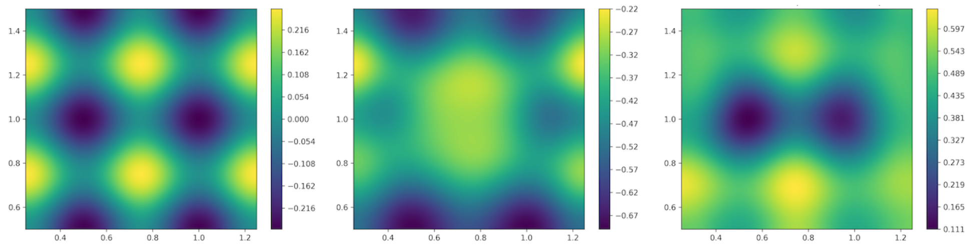
(c) Case C: with noisy velocity and pressure data



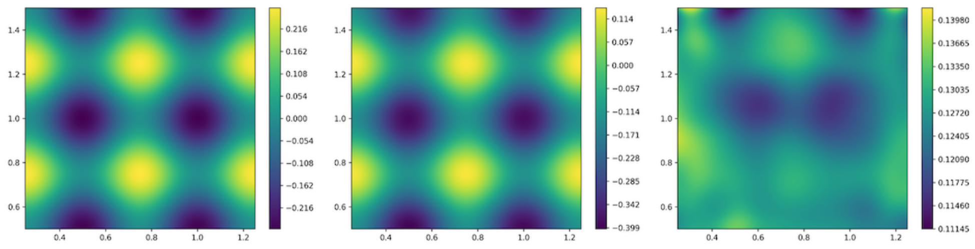
(d) Case D: without boundary conditions

Figure 7.5: Comparison of the velocity u_2 fields for different test cases (a–d) at the same time instant. For each case, the subpanels in order show: the ground truth solution, the predicted solution by the neural network, and the absolute error between the two. Note that the scales differ between the panels.

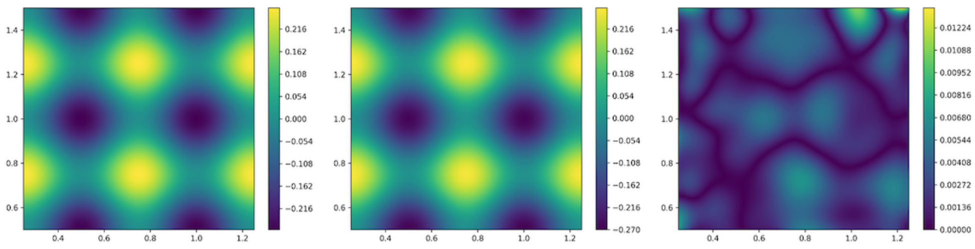
Pressure p



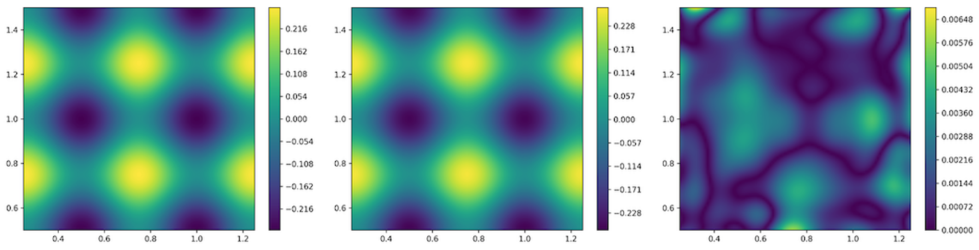
(a) Case A: without data



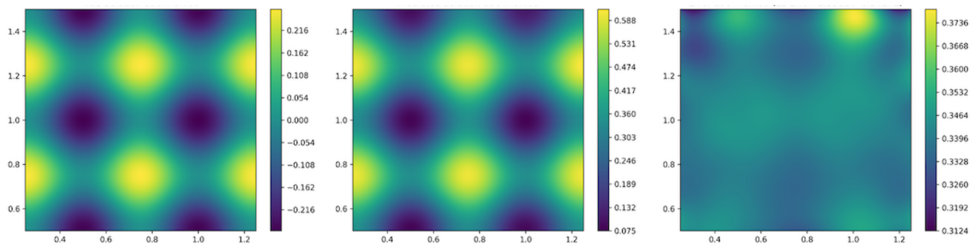
(b) Case B: with velocity data



(c) Case C: with velocity and pressure data



(d) Case D: with noisy velocity and pressure data

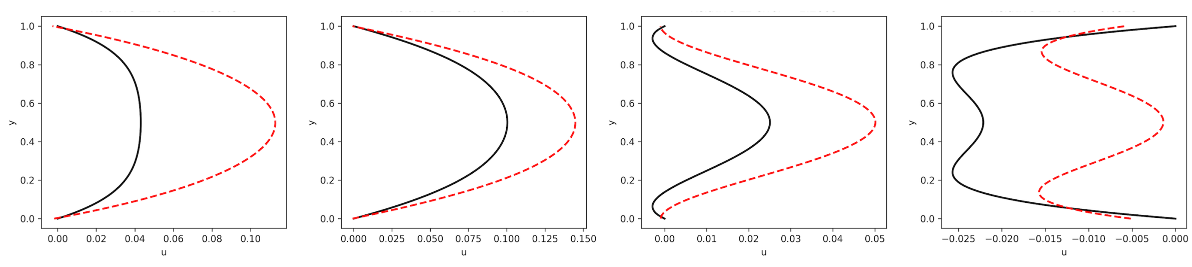


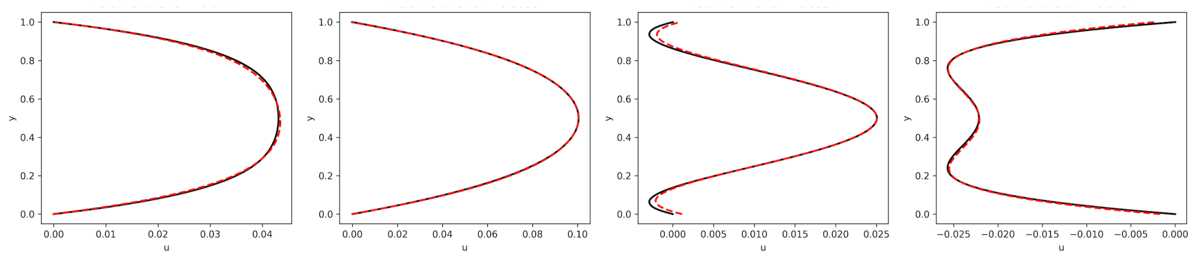
(e) Case E: without boundary conditions

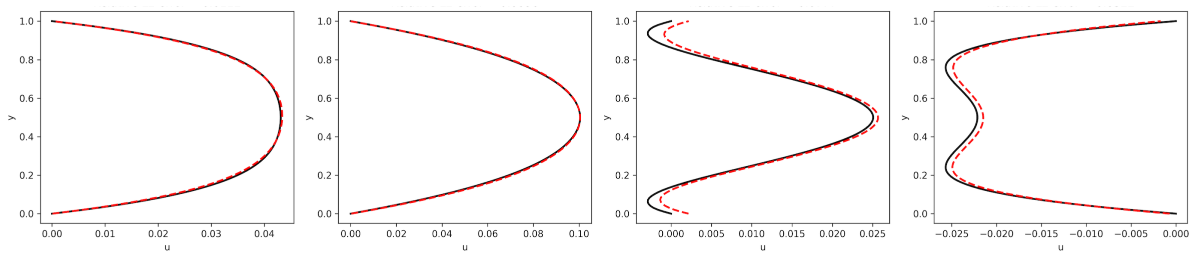
Figure 7.6: Comparison of the pressure fields p for different test cases (a–d) at the same time instant. For each case, the subpanels in order show: the ground truth solution, the predicted solution by the neural network, and the absolute error between the two. Note that the scales differ between the panels.

7.4.5 Womersley Flow

Velocity u


(a) Case A: without data


(b) Case B: with velocity and pressure data


(c) Case C: with noisy velocity and pressure data

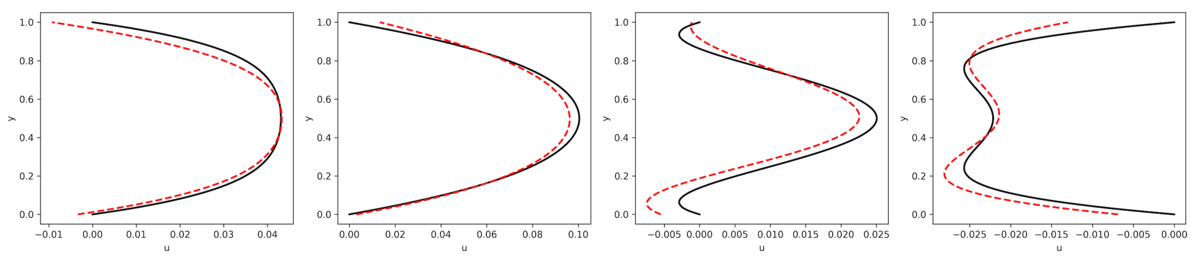
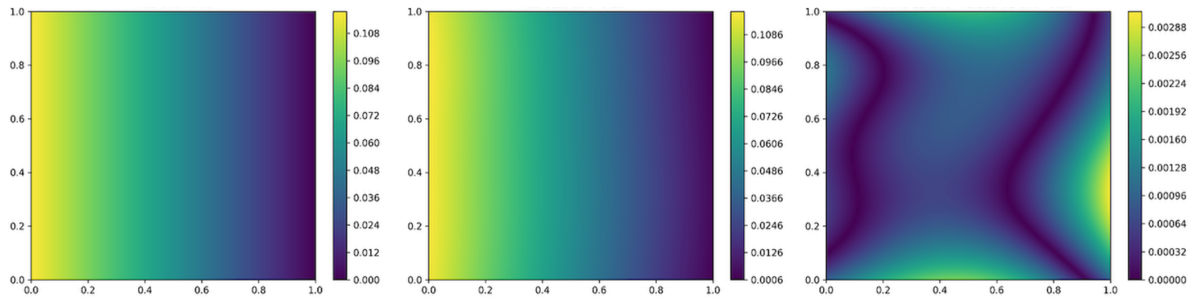
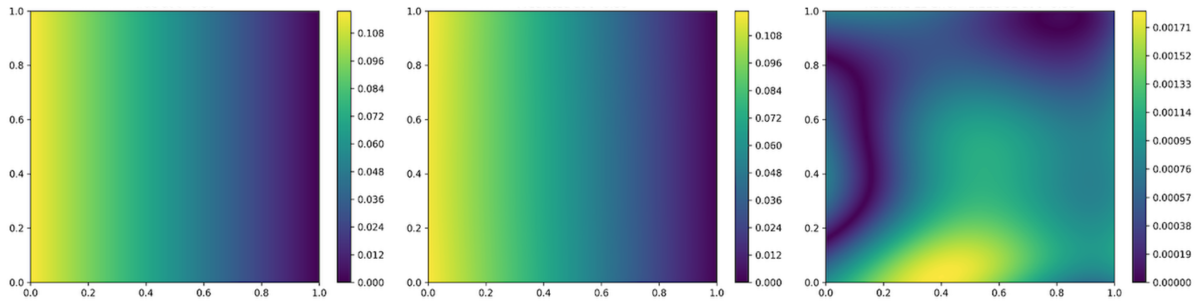

(d) Case D: without boundary conditions

Figure 7.7: Comparison of the velocity field u across different test cases (a–d). For each case, the subpanels show the temporal evolution of the solution at selected time steps (0.4, 0.8, 1.4, and 1.6). The ground truth is shown in black, while the predicted solution is shown in red.

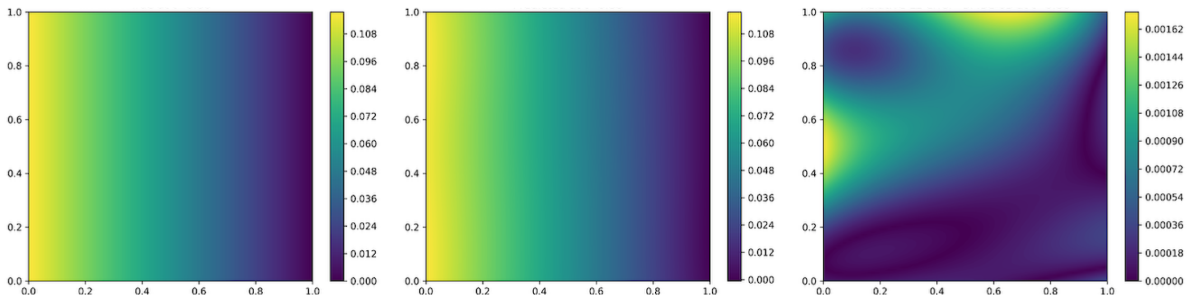
Pressure p



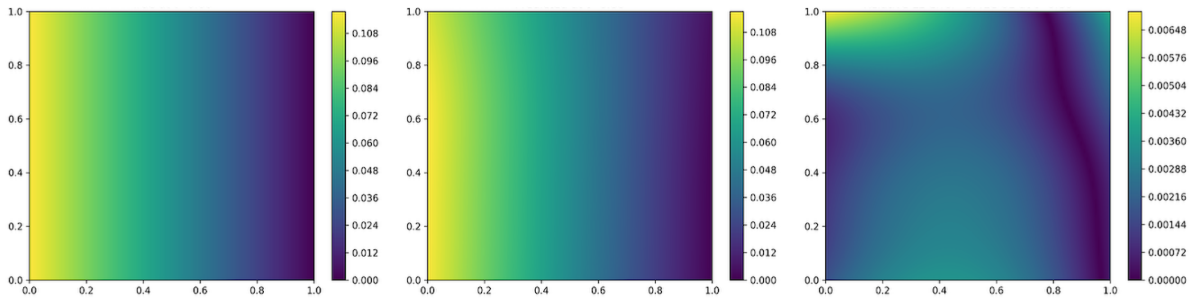
(a) Case A: without data



(b) Case B: with velocity and pressure data



(c) Case C: with noisy velocity and pressure data



(d) Case D: without boundary conditions

Figure 7.8: Comparison of the pressure p fields for different test cases (a-d) at the same time instant. For each case, the subpanels in order show: the ground truth solution, the predicted solution by the neural network, the absolute error between the two, and the comparison between them. Note that the scales differ between the panels.

7.5 Discussion

The following key findings summarize the performance and robustness of the proposed method under various conditions and configurations. The analysis focuses on the impact of data assimilation strategies, noise robustness, boundary condition treatment, and the effects of training parameters.

- The integration of experimental data through data assimilation proved to be the most effective strategy in terms of predictive accuracy of the network, yielding at least an order of magnitude reduction in error compared to approaches without assimilation. No substantial difference emerges between the use of velocity-only data and the combination of velocity and pressure data, which is advantageous from a practical standpoint, since pressure can only be measured invasively. However, it is noteworthy, as highlighted in the Kim&Moin problem, that in the absence of pressure data, or imposed boundary conditions and observational data related to pressure, the field remains defined only up to an additive constant, making the absolute predicted values insignificant.

- The analysis conducted with noisy data highlights the robustness of the method even in realistic scenarios: the increase in error with growing noise standard deviation remains limited.

- Furthermore, the removal of boundary conditions, although accompanied by the inclusion of observed data, does not compromise the network's ability to learn physically consistent solutions, however, it results in a decrease in accuracy, particularly evident in the Womersley problem.

- Finally, the fine-tuning analysis concerning the training duration and collocation point density shows that a higher number of training epochs leads to error reduction, at the expense of increased computational cost. Similarly, increasing the number of collocation points can lead to improvements in solution accuracy but also results in a more substantial increase in computational demand. However, an increase in the number of points does not always lead to error improvement, as in the case of Kim&Moin, where adding more points can lead to the risk of overfitting. Moreover, the distribution of these points, whether random or uniform, has a limited impact on the final accuracy. These results suggest that solution quality depends more on the depth of optimization than on the granularity or arrangement of the spatial discretization.

Chapter 8

Application of a PINN on 3D Geometries: Poiseuille Flow

The Poiseuille equation is a special case derived from the Navier-Stokes equations, where simplified conditions allow for an analytical solution. It describes the laminar flow of an incompressible fluid inside a tube with a constant circular cross-section, assuming the tube's length is much greater than its diameter. This model is widely applied in fields ranging from hemodynamics in blood vessels to fluid transport in pipelines and the design of microfluidic channels.
The flow is unidirectional, meaning the fluid moves solely along the axis of the tube, with zero velocity components in the transverse directions. The axial velocity component, u_x, varies across the cross-section but remains constant along the flow direction, i.e., $\frac{\partial u_x}{\partial x} = 0$. If the pressure at the inlet is P_1 and at the outlet is P_0, a constant pressure gradient $\frac{dP}{dx}$ is established, which drives the motion. Under these conditions, the Poiseuille equation describes the pressure drop due to viscosity, resulting in the parabolic velocity profile characteristic of laminar flow in cylindrical tubes. However, as the velocity increases or the tube diameter becomes larger, the flow may transition to a turbulent regime once a critical threshold defined by the Reynolds number is exceeded. In such cases, the assumptions underlying the Poiseuille equation no longer apply, and the actual pressure losses are greater than those predicted analytically. Furthermore, additional factors such as variations in cross-section, bends, or local resistances can also contribute to pressure losses [105].

8.1 Methods

For this problem, the same network architecture and parameters described in Section 7.1.1 were employed. However, certain adjustments were necessary, such as modifying the number of collocation points and training iterations. Unlike the two cases presented in the previous Chapter, this scenario involves a three-dimensional geometry and a steady-state formulation. Specifically, the neural network takes as input the three spatial coordinates (x, y, z) and outputs four quantities (u, v, w, p), corresponding to the velocity components and pressure

field. For simplicity, the same network was adopted to predict both velocity and pressure fields.
Several experiments were conducted to assess the performance of the model under different configurations: using only the governing PDEs, integrating observational data, and incorporating noisy data. In addition, tests were carried out for different Reynolds numbers by varying the pressure gradient, in order to evaluate the network's robustness across flow regimes. Furthermore, multiple formulations of the Navier–Stokes equations were tested, ranging from the classical and non-dimensional forms to the First-Order NSE-based formulation, to examine their respective impacts on training stability and predictive accuracy.

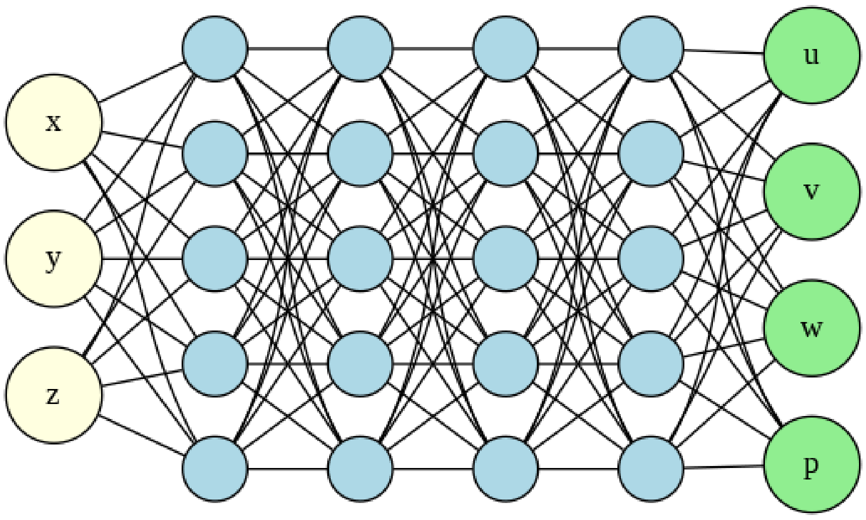


Figure 8.1: Architecture of the Physics-Informed Neural Network. The network consists of three input layers (x, y, z), representing the spatial coordinates, followed by four hidden layers with a fixed number of neurons. Finally, the output layers (u, v, w, p), corresponding to the velocity components and the pressure field.

8.1.1 Governing Equations

The flow is governed by the steady-state Navier-Stokes equations for an incompressible fluid 5.1
Considering different pressure values as boundary conditions, different forms of the Navier–Stokes equations were used. The corresponding Reynolds numbers obtained in the simulations were 0.5, 6.25, and 62.5, depending on the imposed pressure gradient. In particular, for low Reynolds numbers, the classical dimensional Navier–Stokes equations were employed (DNSE), while for higher Reynolds numbers, the non-dimensional form of the Navier–Stokes equations 5.1.1 (ANSE) was adopted to enhance numerical stability and reduce the influence of scale-dependent parameters.
Moreover, additional simulations were carried out by adopting the First-Order NSE formulation of the governing equations. This approach reformulates the

momentum equation by introducing the Cauchy stress tensor 5.1.2 (DNSE). The corresponding non-dimensional formulation of this equation (ANSE) was also considered for higher Reynolds numbers 5.1.3.

8.1.2 Computational Domain

A three-dimensional cylinder along the x-axis, characterized by a radius of 0.5 and a length of 1, was generated using GMSH, an open-source meshing tool. This geometry was selected to represent a simple, idealized model of laminar pipe flow, allowing the study of fluid dynamics under controlled, non-turbulent conditions. The cylinder was discretized into a mesh consisting of $11,917$ nodes, chosen as a trade-off between spatial resolution and computational efficiency. The coordinates of these mesh nodes were then imported into the code and used as input points to train the neural network.

8.1.3 Definition of Boundary Conditions

In this section, we define the boundary conditions used in the simulations, used to enforce velocity and pressure constraints.

Case 1) Different Pressure Drops

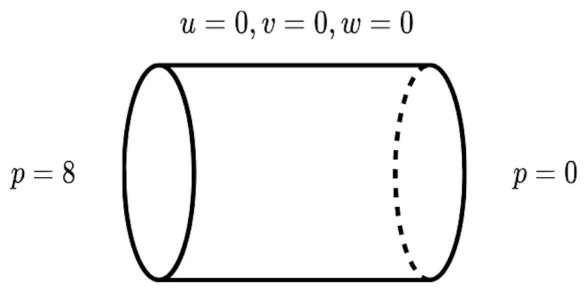


Figure 8.2: Representation of the computational domain used in the Poiseuille simulation. The domain is a cylinder with a radius of 0.5 and a length of 1, where boundary conditions for the pressure are applied.

The boundary conditions include:

- Inlet Boundary Condition: At the inlet of the geometry, a boundary condition has been applied to specify the pressure, setting it to a constant value. This condition is imposed on the boundary where $x = 0$, representing the inlet of the pipe. Several variations of this condition have been tested to assess their impact on the overall flow behavior, considering different pressure values: 8, 100, and 1000.

- Outlet Boundary Condition: At the outlet, the pressure is set to 0, as a reference pressure. This condition is applied on the boundary where $x = 1$, representing the outlet of the pipe.

- Wall Boundary Condition: For the velocity field $\mathbf{u} = (u, v, w)$, no-slip conditions are applied along the domain walls, defined by the equation $y^2 + z^2 = R^2$. Since the pipe is assumed to be rigid, this implies that the velocity on the boundary is zero, i.e., $\mathbf{u} = \mathbf{0}$ on the wall.

Case 2: Different Parabolic Velocity Profiles

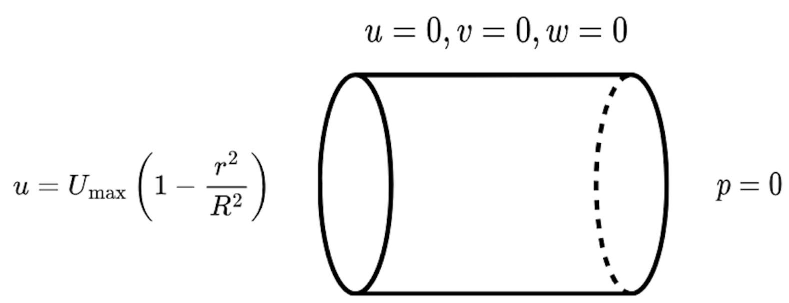


Figure 8.3: Representation of the computational domain used in the Poiseuille simulation. The domain is a cylinder with a radius of 0.5 and a length of 1, where boundary conditions for the velocity are applied.

The boundary conditions include:

- Inlet Boundary Condition: At $x = 0$, a parabolic velocity profile is imposed for the axial velocity component, while the transverse velocity components are set to zero. The axial velocity profile is defined as:

$$u(r) = U_{\max}\left(1 - \frac{r^2}{R^2}\right), \tag{8.1}$$

  where $r^2 = y^2 + z^2$, and R is the radius of the cylinder. Several variations of this condition have been tested to assess their impact on the overall flow behavior, considering different pressure values: 0.5, 6.25, and 62.5.

- Outlet Boundary Condition: The pressure is set to 0, as a reference pressure. This condition is applied on the boundary where $x = 1$, representing the end of the pipe.

- Wall boundaries: No-slip condition is applied for velocity, as in the previous case.

8.1.4 Data Assimilation

The data were derived from the exact solutions of the velocity and pressure fields, with the velocity field modeled using the following analytical expressions for u, v, w, and p:

$$u = \frac{\Delta P}{4\mu L}\left(R^2 - (y^2 + z^2)\right) = U_{max}\left(1 - \frac{r^2}{R^2}\right) \tag{8.2}$$

$$v = 0 \tag{8.3}$$

$$w = 0 \tag{8.4}$$

$$p(x) = \Delta P\,(1 - x)\,, \quad \text{dove } \Delta P = P_{\text{in}} - P_{\text{out}} \tag{8.5}$$

8.1.5 Calculation of the Wall Shear Stress

For the Poiseuille flow problem, the WSS was computed by comparing the exact analytical solution with the predicted one. In this simplified cylindrical case, the WSS is constant along the wall and is given by the analytical expression:

$$\tau_w = \mu \left|\frac{\partial u_x}{\partial r}\right|_{r=R} = \frac{\Delta P}{2L} R \tag{8.6}$$

where u_x is the velocity along the x-axis 8.1.4.
The predicted WSS was obtained during post-processing using automatic differentiation. Specifically, the derivatives of the axial velocity component u_x with respect to the transverse coordinates y and z were computed. The expression used for the predicted WSS is:

$$\tau_w = \mu\left(\frac{\partial u_x}{\partial y}\frac{y}{R} + \frac{\partial u_x}{\partial z}\frac{z}{R}\right) \tag{8.7}$$

8.1.6 Error Analysis

The errors are calculated as reported in Section 7.3.5.

8.2 Results

8.2.1 Training Time

Moving to a three-dimensional geometry, the increased complexity compared to the 2D cases naturally leads to longer training times. For this reason, we explored the adoption of a formulation that is also computationally efficient. The

average training time using the classical Navier–Stokes equations, whether in dimensional or dimensionless form, exceeds one hour. In contrast, by rewriting the Navier–Stokes equations using the First-Order formulation, the average training time is significantly reduced, reaching approximately 15 minutes.
The table below reports the training times for the two configurations, expressed in minutes, considering different cases with varying pressure drops.

Table 8.1: Training Time (minutes) for different pressure drops

Configuration	$\Delta P = 8$	$\Delta P = 100$	$\Delta P = 1000$
Classical NSE	85.14	81.34	82.18
First-Order NSE	15.34	14.28	14.88

The tables below provide a quantitative analysis of the relative l_2 error for velocity along the x-axis (e_u), pressure (e_p) and WSS (e_{WSS}).

8.2.2 Classical NSE Formulation: $\Delta P = 8$

The following results were obtained using the classical form of the Navier–Stokes equations, as given in Equation 5.1. To evaluate the network's ability to handle varying conditions in a three-dimensional geometry, such as the inclusion of data, the presence of noisy data, and the omission of boundary conditions, the case with a pressure drop of $\Delta P = 8$ was selected.
For all the cases considered, 5,000 collocation points were sampled within the domain and 5,000 on the boundary. The training was carried out for a total of 15,000 iterations.

Table 8.2: Relative l_2 Errors - Only PDEs and Boundary Conditions

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Uniform Sampling	0.01305	0.01729	0.01814
Increased Collocation Points, More Training Iterations	0.00733	0.00569	0.00965

Table 8.3: Relative l_2 Errors - Data Assimilation

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Velocity Data (5000 pts)	0.00664	0.00290	0.02582
Velocity/Pressure Data (5000 pts)	0.00519	0.00083	0.01898

Table 8.4: Relative l_2 Errors - Noisy Data Assimilation

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Velocity Data (std = 0.1)	0.00785	0.00344	0.03396
Velocity/Pressure Data (std = 0.1)	0.00643	0.00119	0.00991

Table 8.5: Relative l_2 Errors - Data Assimilation without Boundary Conditions

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Velocity Data (5000 pts)	0.01390	0.77839	0.03622
Velocity/Pressure Data (5000 pts)	0.00706	0.00098	0.02552

8.2.3 Different NSE Formulation: Different Pressure Drops (ΔP)

This section presents the results obtained using both the classical formulation of the Navier–Stokes equations (Equation 5.1) and the First-Order formulation (Equation 5.1.2). For configurations involving higher pressure drop values, the non-dimensional formulation was adopted to improve numerical stability.
In all cases, 5,000 collocation points were sampled within the domain and 5,000 along the boundaries. Additionally, 5,000 velocity and pressure data points were incorporated into the training process. The training was performed over 15,000 iterations.

Table 8.6: Relative l_2 Errors – Pressure Drop ($\Delta P = 8$)

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Classical DNSE	0.00519	0.00083	0.01898
First-Order DNSE	0.00645	0.00099	0.01007

Table 8.7: Relative l_2 Errors – Higher Pressure Drop ($\Delta P = 100$)

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Classical DNSE	1.66991	0.58883	0.99489
Classical ANSE	0.00394	0.00143	0.00210
First-Order ANSE	0.00422	0.00223	0.01151

Table 8.8: Relative L_2 Errors – Higher Pressure Drop ($\Delta P = 1000$)

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Classical DNSE	1.08142	0.81554	0.99856
Classical ANSE	0.00875	0.00154	0.01013
First-Order ANSE	0.02253	0.00833	0.07910

8.2.4 Different NSE Formulation: Parabolic Velocity Profile

This section presents the results obtained using both the classical form of the Navier–Stokes equations (Equation 5.1) and the First-Order formulation (Equation 5.1.2). For higher parabolic velocity profiles, the non-dimensional formulation was adopted. For all cases, velocity and pressure data were included in the training process.

Table 8.9: Relative l_2 Errors - Parabolic Velocity Profile ($U_{max} = 0.5$)

Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Classical NSE	0.00712	0.00104	0.00811
First-Order DNSE	0.00402	0.00263	0.00979

Table 8.10: Relative l_2 Errors - Parabolic Velocity Profile ($U_{max} = 6.25$)

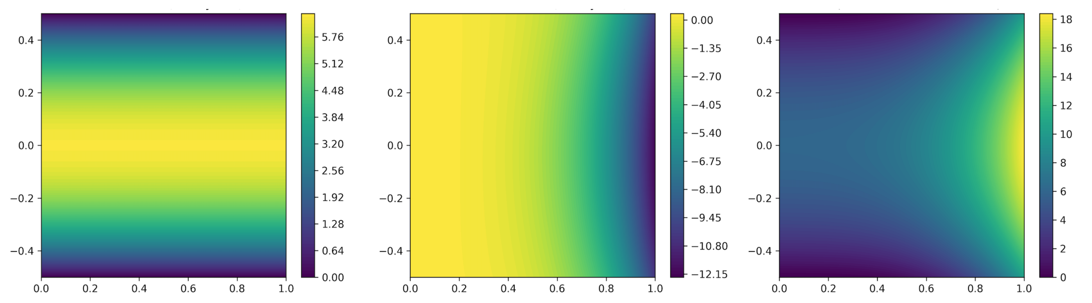
Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Classical ANSE	0.00153	0.00088	0.00248
First-Order ANSE	0.00405	0.00204	0.01292

Table 8.11: Relative l_2 Errors - Parabolic Velocity Profile ($U_{max} = 62.5$)

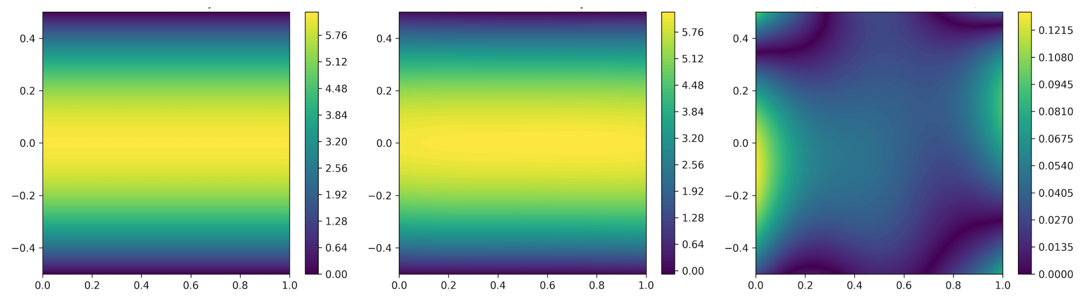
Configuration	e_u **(Velocity)**	e_p **(Pressure)**	e_{WSS} **(WSS)**
Classical ANSE	0.00249	0.00230	0.00513
First-Order ANSE	0.01797	0.06872	0.01124

The following figures show the velocity and pressure fields, as well as the WSS, considering different Navier-Stokes formulations. For each case, the subpanels are arranged in the following order: the ground truth solution, the predicted solution by the neural network, and the absolute error between the two. As an illustrative example, only the results corresponding to the configuration with a pressure drop of 100 imposed as the inlet boundary condition are reported.

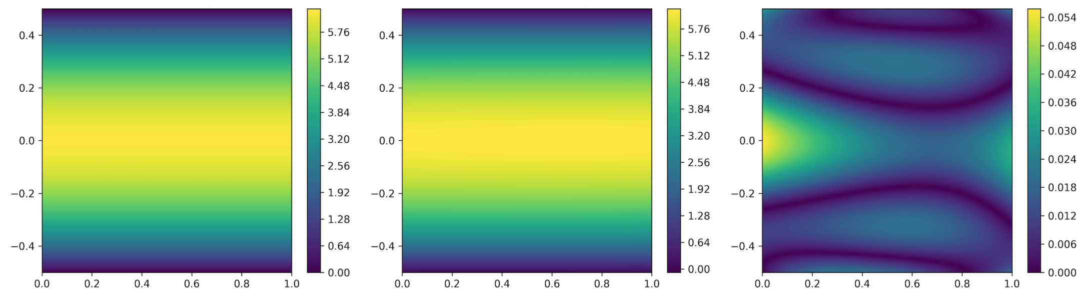
Velocity u



(a) Case A: Dimensional NSE



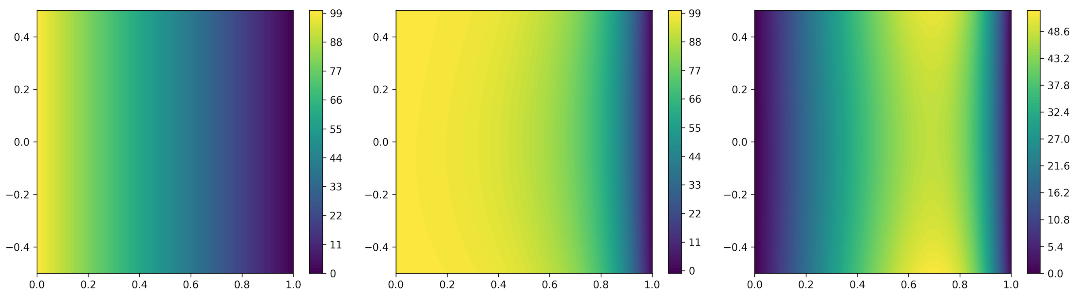
(b) Case B: Non-Dimensional NSE



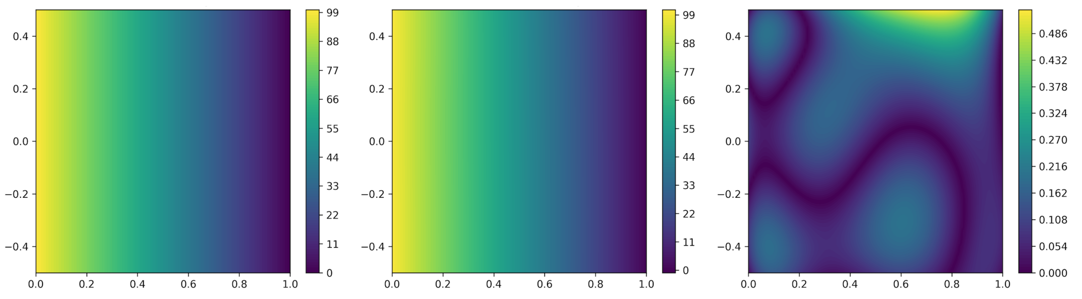
(c) Case C: First-Order NSE

Figure 8.4: Comparison of the velocity fields for a pressure drop of 100 (a–c), considering the different formulation of the Navier-Stokes Equations. Note that the scales differ between the panels.

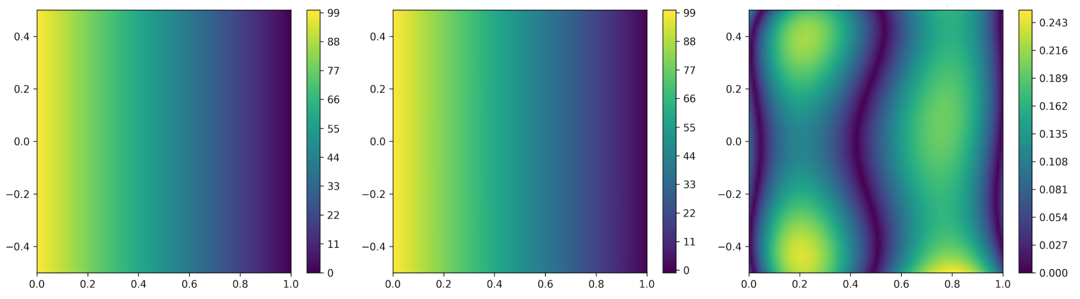
Pressure p



(a) Case A: Dimensional NSE



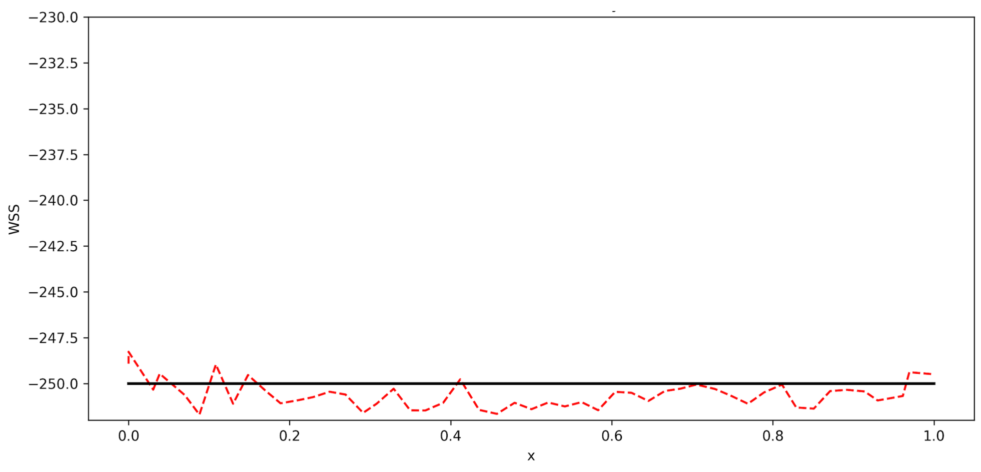
(b) Case B: Non-Dimensional NSE



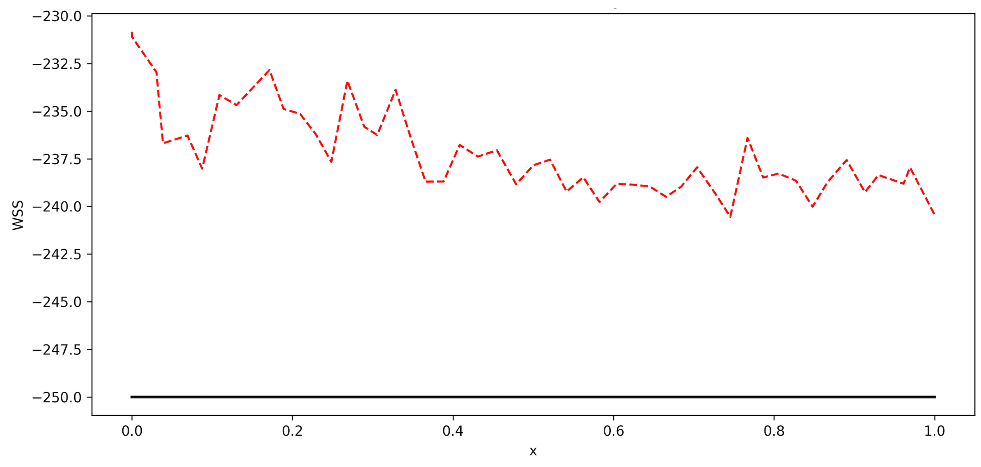
(c) Case C: First-Order NSE

Figure 8.5: Comparison of the pressure field for a pressure drop of 100 (a–c), considering the different formulation of the Navier-Stokes Equations. Note that the scales differ between the panels.

Wall Shear Stress



(a) Case A: Non-Dimensional NSE



(b) Case B: First-Order NSE

Figure 8.6: Comparison of the WSS for a pressure drop of 100 (a–b), using different formulations of the Navier–Stokes equations. The ground truth is shown in black, and the predicted solution in red.

8.3 Discussion

In the transition to three-dimensional analysis, the observations made in the two-dimensional context are confirmed.

- Having data in the loss function makes the PINN much more accurate, showing no substantial differences between the use of velocity data alone and the combination with pressure data.
- The algorithm also demonstrates robustness in the presence of noisy data, maintaining stable and accurate predictions. Furthermore, the increase in error remains limited even as the noise level increases, confirming the method's effectiveness under realistic conditions.

- Even without the explicit imposition of boundary conditions, the network is still able to reconstruct a physically consistent solution.

- Different boundary condition configurations were also analyzed: the model effectively handled both pressure conditions and parabolic velocity profiles at the inlet. For moderate values, such as a pressure drop of 8 or a maximum velocity of 0.5, the classical Navier–Stokes formulation performed adequately without convergence issues. However, in cases involving higher pressure drops or inlet velocities, which cause significant imbalances in the loss function terms, *the non-dimensional formulation proved to be more stable and effective*. In these scenarios, the dimensional formulation struggled and often failed to converge to a solution. When using the non-dimensional formulation, the resulting errors remained comparable to those obtained under lower pressure and velocity conditions. This suggests that, as long as the problem is well-posed, the Reynolds number does not substantially affect the network's ability to converge to an accurate solution.

- Lastly, the First-Order formulation showed computational advantages: with the same architecture, number of iterations, and problem setup, it enables a reduction in training time of *up to one-fifth* compared to the classical formulation. This approach proved effective for both moderate and high values of pressure and velocity, provided the non-dimensional form is adopted in the latter case. However, the gain in efficiency comes at the cost of a slight increase in error, which remains limited. This error in the velocity field also affects the calculation of the WSS, which is more accurately estimated when using the classical formulation.

Chapter 9

Towards Semi-Realistic Models of Vascular Geometries: Curved Tube

Modeling blood flow in realistic vascular geometries is a challenging task due to the complex three-dimensional structures of arteries, especially their curvature and branching patterns. In this chapter, we explore the application of PINNs to simulate flow in curved geometries that resemble human coronary arteries These vessels exhibit pronounced curvature as they follow the surface of the myocardium, with the left main coronary artery having an approximate diameter of 4 mm, and the left anterior descending and circumflex branches measuring around 3 mm [106].
To accurately capture the hemodynamics within these curved vessels, we focus on key dimensionless parameters, among which the curvature ratio is fundamental. It is defined as:

$$d = \frac{a}{R} \tag{9.1}$$

where a denotes the vessel radius and R the mean radius of curvature. In the left coronary artery tree, the curvature ratio d varies significantly depending on the anatomical location, typically ranging from 0.02 to 0.5.
In addition, the Reynolds number Re, which quantifies the ratio of inertial to viscous forces, depends on physiological parameters such as heart rate and blood flow velocity, and plays a key role in characterizing the flow regime.
To capture the combined effect of the curvature and the Reynolds number, the Dean number k is introduced [107], defined as:

$$k = \sqrt{d} \cdot Re \tag{9.2}$$

The Dean number quantifies the ratio of centrifugal and inertial forces to viscous forces in curved ducts, and plays a key role in determining the development of secondary flow structures. At low Dean numbers, typically below 40-60, the flow remains predominantly unidirectional and exhibits negligible secondary motion. As the Dean number increases to intermediate values around 60–75, secondary flow begins to emerge within the cross-sectional plane, often manifesting as symmetric vortical structures. When the Dean number exceeds approximately

75–200, a secondary instability arises, characterized by increasingly complex dynamics such as vortex undulations, twisting, merging, and eventually vortex pair splitting [108].
In this study, both the Reynolds number and the curvature ratio were systematically varied in order to evaluate their combined influence, via the Dean number, on flow behavior. This framework serves to investigate the capability of PINNs to accurately capture the dynamics arising in curved vascular geometries, and to assess their robustness in handling increasingly complex hemodynamic regimes.

9.1 Methods

For this problem, the same network architecture and parameters described in Section 7.1.1 were employed. However, certain adjustments were necessary, such as modifying the number of collocation points and training iterations. The neural network takes as input the three spatial coordinates (x, y, z) and outputs four quantities (u, v, w, p), corresponding to the velocity components and pressure field. For simplicity, the same network was adopted to predict both velocity and pressure fields.

9.1.1 Governing Equations

For the problems considered, the flow is governed by the non-dimensional steady-state Navier–Stokes equations for an incompressible fluid 5.1.1. Furthermore, some tests have also been conducted using the First-Order formulation 5.1.3.
To investigate the influence of flow regime, simulations were performed for different Reynolds numbers, specifically $Re = 10, 100$, and 300, by varying the imposed pressure gradient or the maximum velocity of the parabolic profile. The kinematic viscosity was assumed constant and equal to 4 mm^2/s [109].

9.1.2 Computational Domain

For the numerical experiments, different 3D geometries were considered, each representing a curved tube. The geometries for the cases under study were generated and defined using the GMSH software, followed by mesh creation. The mesh nodes were then imported and used as training points for the neural network.

Geometry 1

In this case, the geometry has a minor radius of 4 mm and a major radius of 133.33 mm, resulting in a curvature ratio $d = 0.03$. The domain was discretized using a fine mesh comprising 37,509 nodes.

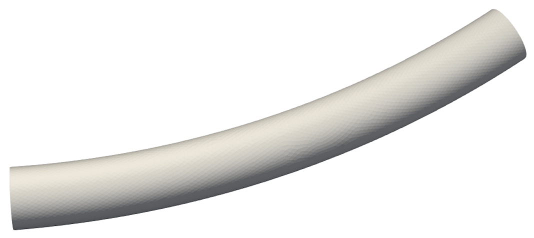


Figure 9.1: Representation of the computational domain for Case 1.

Geometry 2

In this case, the geometry has a minor radius of 4 mm and a major radius of 50 mm, resulting in a curvature ratio $d = 0.08$. The domain was discretized using a fine mesh comprising 35,930 nodes.

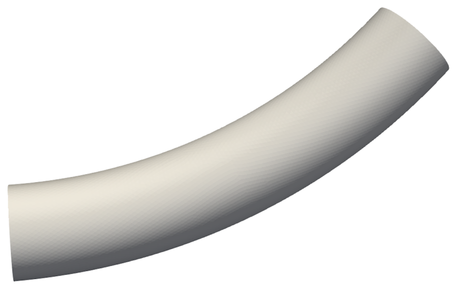


Figure 9.2: Representation of the computational domain for Case 2.

Geometry 3

Finally, in the third case, the geometry has a minor radius of 4 mm and a major radius of 33.333 mm, resulting in a curvature ratio $d = 0.12$. The domain was discretized using a fine mesh comprising $32,572$ nodes.

9.1.3 Definition of Boundary Conditions

Two types of inlet boundary conditions were considered: one based on an imposed pressure drop, and the other on a prescribed parabolic velocity profile.
For all the cases considered, the inlet pressure and parabolic velocity profiles were imposed as *soft constraints*, by minimizing the corresponding residual in the loss function. On the other hand, no-slip Dirichlet boundary conditions for the velocity field were enforced as *hard constraints*, ensuring that the velocity at the walls was exactly zero, as described in Section 5.2. To implement this condition, a distance-based scaling function was defined and applied to the predicted velocity

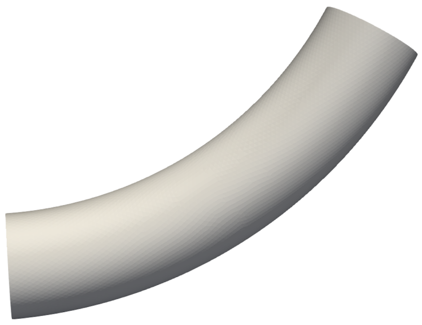


Figure 9.3: Representation of the computational domain for Case 3.

field as follows:

$$\mathbf{u}_{\text{new}} = \left(\left(\sqrt{X^2 + Y^2} - R \right)^2 + Z^2 - r^2 \right) \cdot \mathbf{u}_{\text{pred}}, \tag{9.3}$$

where R and r represent the major radius and the minor radius of the geometry, respectively.

Case 1) Different Pressure Drops

- Inlet Boundary Condition: At the inlet ($x = 0$), a fixed pressure value is prescribed based on the specific case configuration. These values vary according to the Reynolds number, as shown in Table 9.1.
- Outlet Boundary Condition: A zero-pressure condition is enforced at the outlet boundary.

Table 9.1: Geometric configurations and different imposed pressure drops for different Reynolds numbers.

Configuration	$Re = 10$	$Re = 100$	$Re = 300$
Geometry 1	350	3600	11225
Geometry 2	200	2090	7150
Geometry 3	180	1940	7035

Case 2) Different Parabolic Velocity Profile

- Inlet Boundary Condition: A parabolic velocity profile was imposed at the inlet boundary ($x = 0$), with the maximum velocity $U_{\max}$ varying according to the Reynolds number. The selected values were $U_{\max} = 5$, 50, and 150, corresponding to Reynolds numbers of 10, 100, and 300, respectively.

  The parabolic profile is defined as follows:

$$u = U_{\max} \left(1 - \frac{(y - R)^2 + z^2}{r^2} \right)$$

- Outlet Boundary Condition: A zero-pressure condition is enforced at the outlet boundary.

9.1.4 Data Assimilation

Computational Fluid Dynamics simulations were performed on the generated geometries using the Finite Element Method. These simulations served both as a reference to evaluate the solutions provided by PINNs and as a source to extract measurement points for data assimilation. The velocity and pressure data obtained from CFD simulations, sampled at mesh nodes, were treated as the reference solution and incorporated into the loss function in the form of residuals to be minimized. For each case considered, approximately $5,000$ points were sampled.

9.1.5 Calculation of The Wall Shear Stress

The WSS was computed from both the CFD reference solution and the PINNs-predicted solution. The software ParaView was employed to evaluate shear stresses on the vessel walls, based on the gradients of the velocity field.

9.1.6 Error Analysis

The errors are calculated as reported in Section 7.3.5.

9.2 Results

9.2.1 Training Time

As observed in the previous chapter, the complexity increases when moving to a three-dimensional geometry. In this context, the average training time using the non-dimensional classical Navier–Stokes equations exceeds one hour, with a typical duration of approximately 75 minutes. In contrast, reformulating the Navier–Stokes equations using the First-Order formulation significantly reduces the training time to around 10–15 minutes.

9.2.2 Results Obtained with the Classical Navier–Stokes Formulation

The tables below provide a quantitative analysis of the relative l_2 error for velocity components (e_u) and (e_v), pressure (e_p), calculated by comparing the predicted solutions with the reference data obtained from the classical Navier–Stokes formulation.

9.2.3 Case 1) Different Pressure Drops (ΔP)

Table 9.2: Relative l_2 Errors – Geometry 1: $d = 0.3$

Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.00887	0.01180	0.01581
$Re = 100$	0.02853	0.06195	0.10544
$Re = 300$	0.12135	0.19626	0.19322

Table 9.3: Relative l_2 Errors – Geometry 2: $d = 0.08$

Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.01189	0.02280	0.01308
$Re = 100$	0.03711	0.07161	0.08726
$Re = 300$	0.06889	0.16264	0.07840

Table 9.4: Relative l_2 Errors – Geometry 3: $d = 0.12$

Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.01988	0.03078	0.02413
$Re = 100$	0.03722	0.06712	0.08445
$Re = 300$	0.09603	0.20724	0.04430

9.2.4 Case 2) Different Parabolic Velocity Profile

Table 9.5: Relative l_2 Errors – Geometry 1: $d = 0.03$

Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.04516	0.05088	0.00328
$Re = 100$	0.01792	0.05085	0.01371
$Re = 300$	0.11092	0.19533	0.04398

Table 9.6: Relative l_2 Errors – Geometry 2: $d = 0.08$

Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.06735	0.08685	0.00697
$Re = 100$	0.02418	0.05966	0.03371
$Re = 300$	0.07221	0.15969	0.10684

Table 9.7: Relative l_2 Errors – Geometry 3: $d = 0.12$

Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.06703	0.08322	0.00707
$Re = 100$	0.02295	0.05312	0.05615
$Re = 300$	0.07589	0.17788	0.10055

9.2.5 Results Obtained with the First Order Navier–Stokes Formulation

The tables below provide a quantitative analysis of the relative l_2 error for velocity components (e_u) and (e_v), pressure (e_p), calculated by comparing the predicted solutions with the reference data obtained from the First-Order NSE formulation, considering a pressure drop imposed at the inlet.

Table 9.8: Relative l_2 Errors – Geometry 3: $d = 0.08$

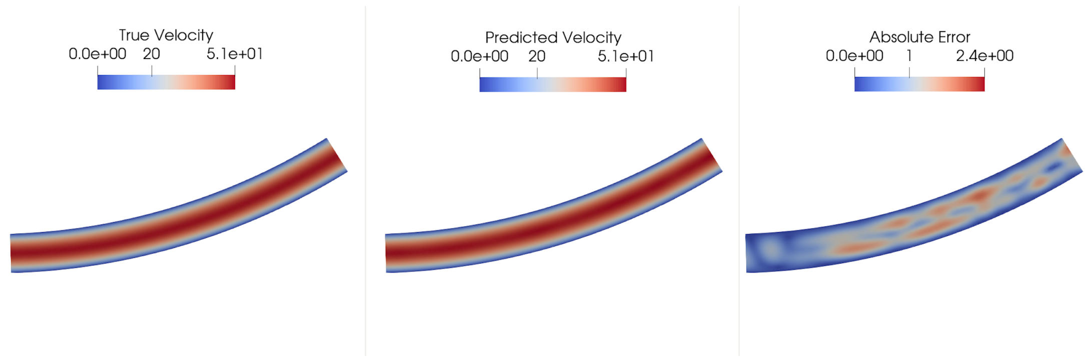
Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$Re = 10$	0.02990	0.02992	0.00831
$Re = 100$	0.01844	0.04928	0.03379
$Re = 300$	0.06690	0.15734	0.08214

Table 9.9: Relative l_2 Errors – $Re = 100$

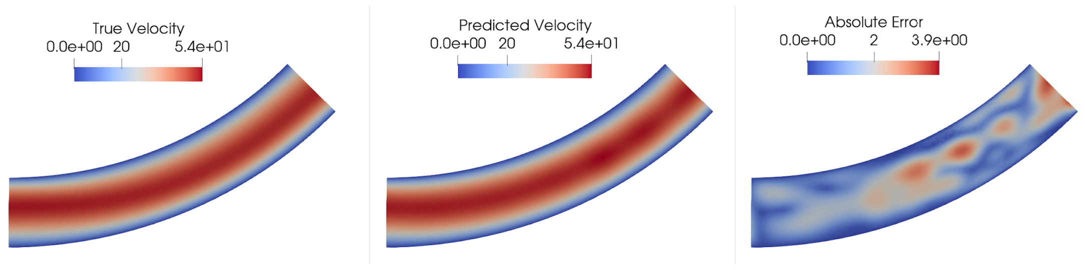
Configuration	e_u (Velocity x)	e_y (Velocity y)	e_p (Pressure)
$d = 0.03$	0.02377	0.06318	0.01116
$d = 0.08$	0.01844	0.04928	0.03379
$d = 0.12$	0.02327	0.05261	0.05084

The subsequent figures present the velocity field, along with the WSS. For each configuration, the subpanels are organized as follows: the ground truth solution, the neural network prediction, and the corresponding absolute error. As a representative case, results at a Reynolds number of 100 are reported for all three geometries, under an imposed pressure drop at the inlet, with the aim of emphasizing the influence of the curvature ratio.

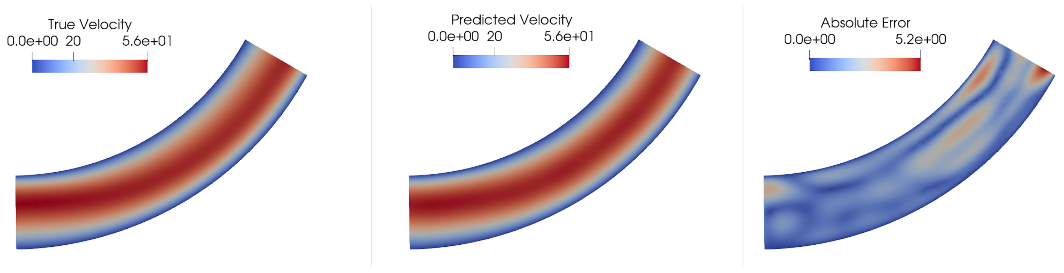
Velocity u - Classical NSE Formulation



(a) Case A: Geometry 1 ($d = 0.03$)



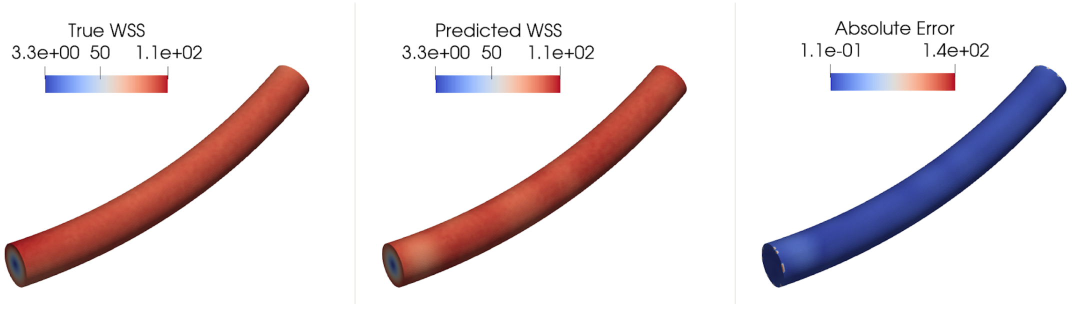
(b) Case B: Geometry 2 ($d = 0.08$)



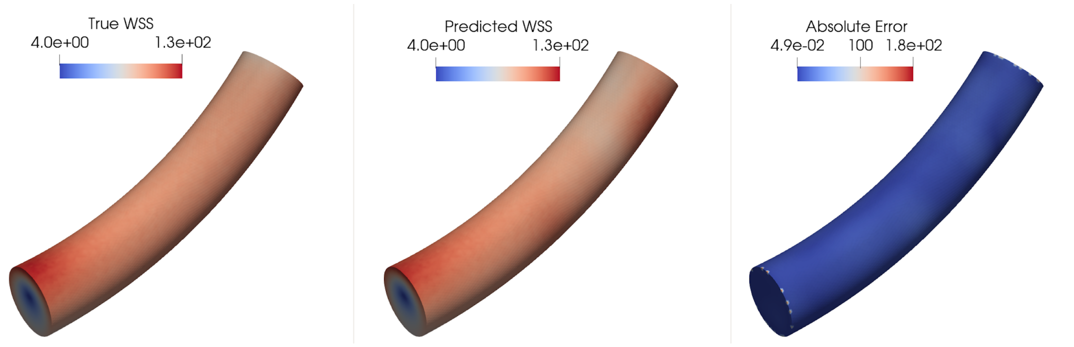
(c) Case C: Geometry 3 ($d = 0.12$)

Figure 9.4: Comparison of the velocity fields for different geometries (a–c) under the application of a pressure drop at Reynolds number $Re = 100$. Note that the scales differ between the panels.

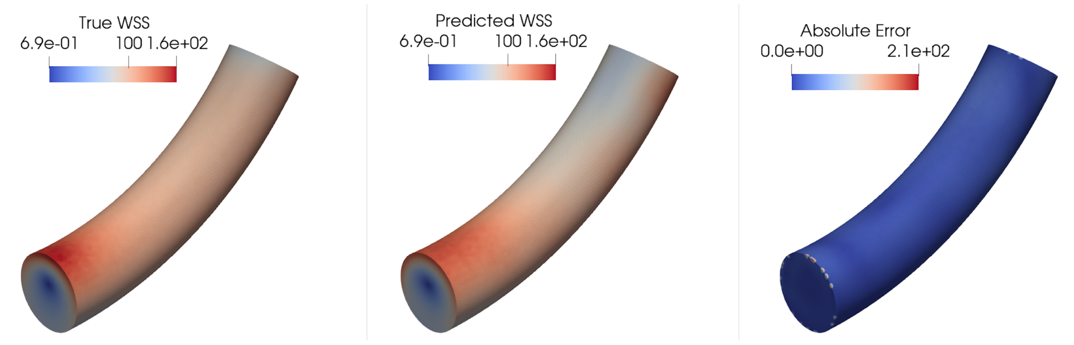
Wall Shear Stress - Classical NSE Formulation



(a) Case A: Geometry 1 ($d = 0.03$)



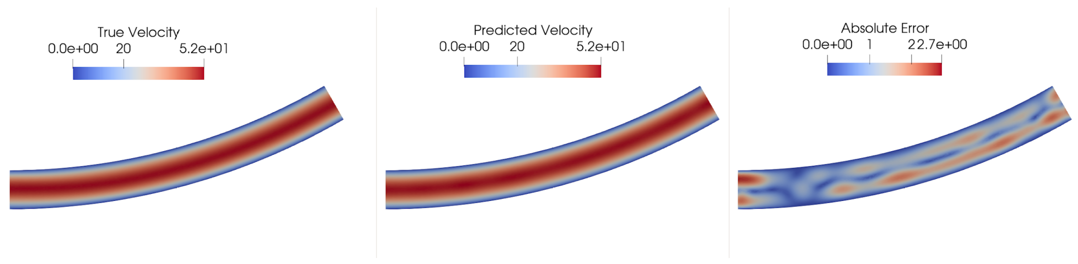
(b) Case B: Geometry 2 ($d = 0.08$)



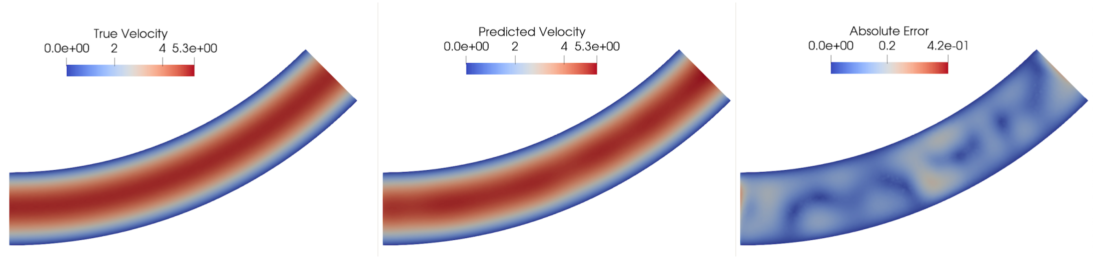
(c) Case C: Geometry 3 ($d = 0.12$)

Figure 9.5: Comparison of the WSS for different geometries (a–c) under the application of a pressure drop at Reynolds number $Re = 100$. Note that the scales differ between the panels.

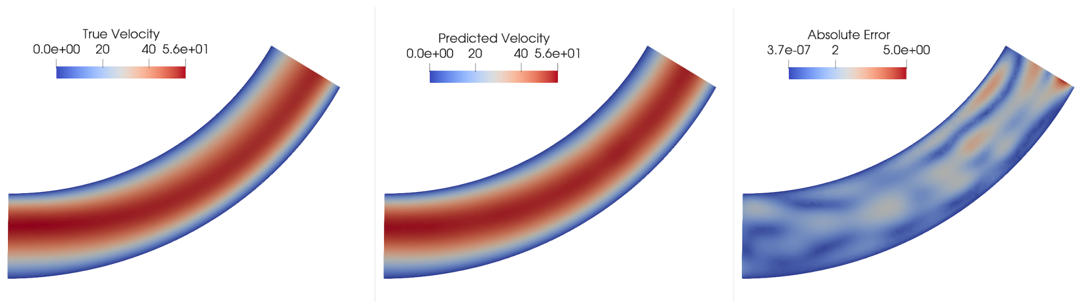
Velocity u - First-Order NSE Formulation



(a) Case A: Geometry 1 ($d = 0.03$)



(b) Case B: Geometry 2 ($d = 0.08$)

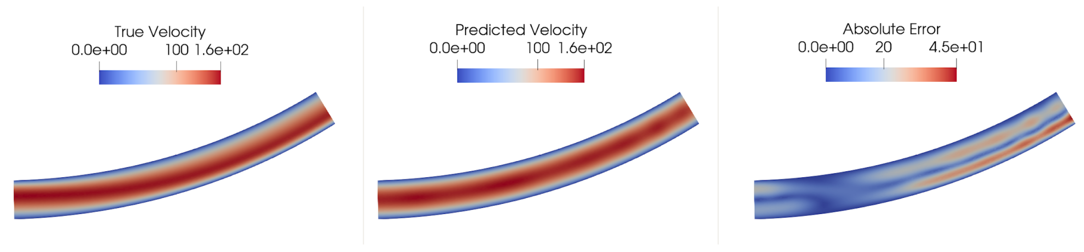


(c) Case C: Geometry 3 ($d = 0.12$)

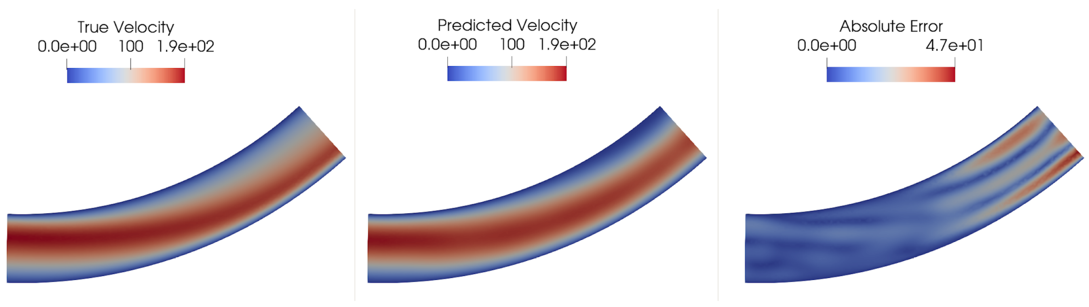
Figure 9.6: Comparison of the velocity fields for different geometries (a–c) under the application of a pressure drop at Reynolds number $Re = 100$. Note that the scales differ between the panels.

The subsequent figures present the velocity field, along with the WSS. For each configuration, the subpanels are organized as follows: the ground truth solution, the neural network prediction, and the corresponding absolute error. As a representative case, results at a Reynolds number of 300 are reported for all three geometries, under an imposed pressure drop at the inlet, with the aim of assessing the model's behavior at higher Reynolds numbers.

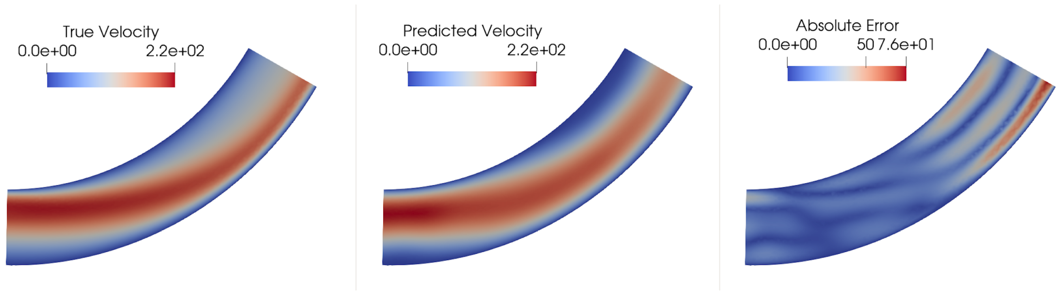
Velocity u - Classical NSE Formulation



(a) Case A: Geometry 1 ($d = 0.03$)



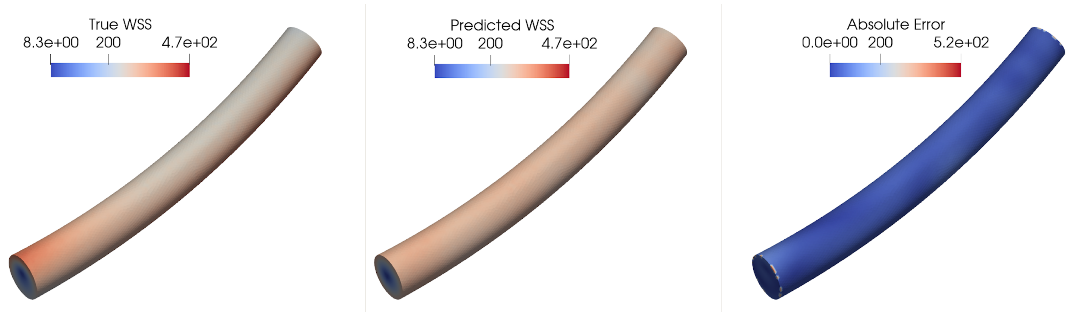
(b) Case B: Geometry 2 ($d = 0.08$)



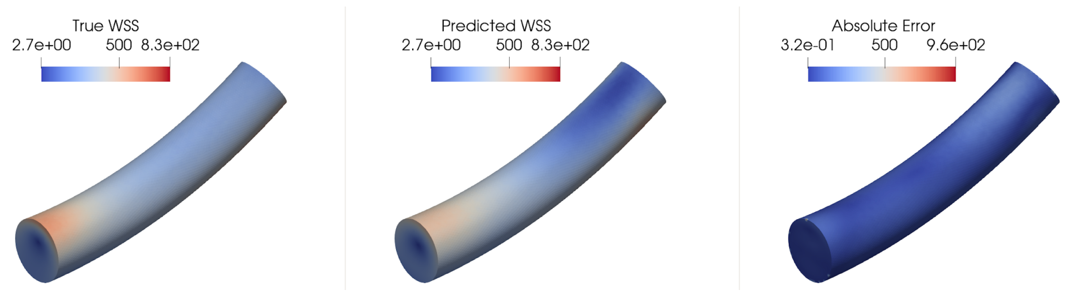
(c) Case C: Geometry 3 ($d = 0.12$)

Figure 9.7: Comparison of the velocity fields for different geometries (a–c) under the application of a pressure drop at Reynolds number $Re = 300$. Note that the scales differ between the panels.

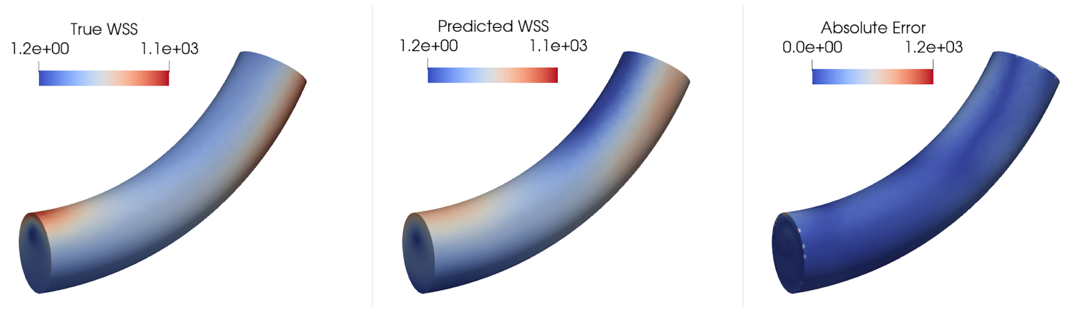
Wall Shear Stress - Classical NSE Formulation



(a) Case A: Geometry 1 ($d = 0.03$)



(b) Case B: Geometry 2 ($d = 0.08$)



(c) Case C: Geometry 3 ($d = 0.12$)

Figure 9.8: Comparison of the WSS for different geometries (a–c) under the application of a pressure drop at Reynolds number $Re = 300$. Note that the scales differ between the panels.

9.2.6 The Importance of Defining Boundary Conditions as *Hard Constraints* - Classical NSE Formulation

The following figure shows the velocity field for Geometry 1 (Case 1) at a Reynolds number of 10, with an imposed pressure drop at the inlet. Here, we aim to demonstrate the impact of boundary condition enforcement by comparing three cases: the true solution, the prediction using soft boundary conditions, and the prediction using hard boundary conditions.

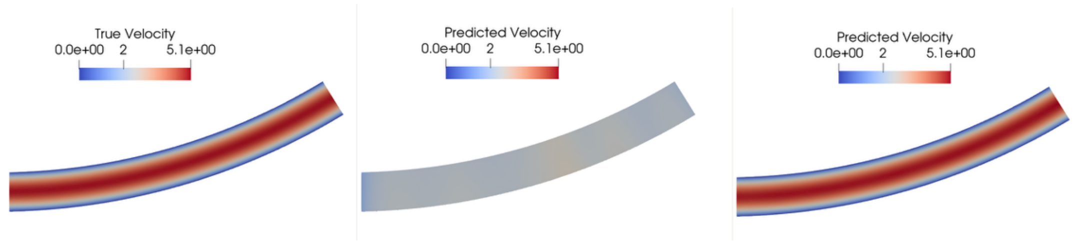


Figure 9.9: Velocity field for Geometry 1 ($d = 0.03$) at Re = 10: reference solution, prediction with soft BCs, and prediction with hard BCs.

9.3 Discussion

The results presented in this work highlight several key aspects of the proposed approach. The following points summarize the most relevant findings:

- The flow analysis in the curved pipe demonstrates strong agreement with reference data for velocity, pressure, and WSS across all examined cases. Notably, simulations using a pressure drop as the inlet condition yield results that closely match those obtained with an imposed parabolic velocity profile, confirming the robustness of the model with respect to different inlet configurations.

- Furthermore, no-slip boundary conditions must be imposed strongly, that is, explicitly enforced as well-defined constraints, rather than softly incorporated into the loss function. This highlights the critical importance of accurately defining boundary conditions to ensure both the physical consistency and the predictive accuracy of the model.

- When comparing the two formulations, the classical Navier–Stokes equations and the First-Order NSE, similar error levels are observed, while the First-Order formulation offers a significant reduction in training time. Despite this advantage, further investigation is needed to fully assess its performance and limitations.

- The study considered two fundamental parameters: the curvature ratio and the Reynolds number. Regarding the curvature ratio, no significant variations in the predictive performance of the model were observed, as the estimated errors remained substantially unchanged when varying this parameter. Conversely, the Reynolds number impacts prediction accuracy. Estimation errors progressively increase with higher Reynolds numbers due to the growing complexity of the flow, characterized by a disturbed regime and nonlinear phenomena. The largest errors are typically observed near the outlet region. Nevertheless, the model successfully captures the overall flow behavior. In particular, at elevated Reynolds and Dean numbers, flow patterns with pronounced asymmetry arise. The neural network effectively recognizes and approximates these asymmetric configurations, demonstrating strong learning and adaptability to complex flow conditions.

Chapter 10

Conclusions and Future Developments

To conclude, PINNs offer several advantages in the modeling of fluid dynamics.

- In the context of the problems addressed in this work, they demonstrate the ability to accurately predict key physical quantities, such as velocity and pressure fields, as well as WSS, in both steady and unsteady regimes.
- A further significant advantage is observed when data assimilation is performed: as initially hypothesized, the predictive accuracy of PINNs improves notably, whether using velocity data alone or a combination of velocity and pressure data. This is particularly beneficial from a practical standpoint, as pressure can only be measured invasively and is therefore generally not available. Furthermore, PINNs exhibit strong robustness to data quality, effectively handling sparse or noisy datasets while maintaining stable and reliable predictions. This makes the method particularly suitable for clinical contexts, where the available data are often affected by noise. This behavior is consistent with expectations, given that the data assimilation term is introduced through a least-squares formulation, which inherently promotes stability.
- Moreover, in relatively simple geometrical configurations, it is sometimes possible to omit the explicit enforcement of boundary conditions without significantly affecting the accuracy of the solution, thereby simplifying the overall problem setup and reducing computational costs.

Despite these significant advantages, PINNs also present currently several limitations, especially when applied to complex flow scenarios.

- As the problem complexity increases, often due to intricate geometries, the explicit enforcement of boundary conditions becomes essential. In other words, boundary conditions must be imposed as hard constraints rather than integrated into the loss function. Therefore, although PINNs do not rely on a traditional mesh, they still require a sufficiently dense distribution of collocation points to accurately capture the geometry and the domain of interest. As a consequence, some form of meshing or careful sampling remains necessary, especially in cases involving irregular boundaries or complex geometric features.

- Another significant challenge lies in model calibration: fine-tuning PINNs for a specific problem requires careful consideration of several factors. Some of these include the neural network architecture, the choice of hyperparameters, and the formulation of the governing equations. This tuning process is often time-consuming, and the resulting configurations may not generalize well across different problems, frequently necessitating extensive empirical testing to adapt the model to new scenarios.

  In particular, different formulations of the Navier–Stokes equations were considered, each selected based on the specific characteristics of the problem and offering distinct advantages. As for the network architecture, the same model was used to approximate both velocity and pressure fields. While this choice yields accurate results and simplifies implementation, it differs from traditional methods such as FEM, where velocity and pressure are typically approximated in different functional spaces in order to satisfy the *inf-sup condition.* However, since a rigorous theoretical framework for such conditions in the PINNs setting is not yet established, a unified architecture was adopted for simplicity.

10.1 Short-Term Advancements

The work carried out so far, although promising, represents only the first step towards the development of a more advanced research line. In this initial phase, various problems have been addressed, starting from simplified geometries created ad hoc and progressing to configurations of increasing complexity, more closely resembling anatomical structures, such as blood vessels.
The next step involves the use of real vascular structures, which can be obtained through advanced imaging techniques. Among these, 4D Magnetic Resonance Imaging (4D-MRI) stands out, as it enables both the reconstruction of the three-dimensional anatomical domain and the complete acquisition and analysis of blood flow dynamics. 4D-MRI can provide estimates of advanced hemodynamic parameters, such as WSS, pulse wave velocity, and pressure gradients; however, these estimates are not highly precise due to intrinsic limitations in spatial and temporal resolution, as well as measurement noise [110]. A complementary technique is Ultrasound Doppler Velocimetry, which enables the measurement of blood flow velocity propagation [111].
These imaging modalities provide rich and high-resolution datasets, which can be used as input for the next major objective: employing PINNs to solve the Navier–Stokes equations through data assimilation. By integrating experimental data from 4D-MRI or Doppler ultrasound directly into the training process, PINNs offer the potential to model hemodynamics within patient-specific vascular geometries. It will therefore be essential to assess whether, even in such

complex domains, PINNs can produce quantitatively reliable results comparable to those obtained through traditional Computational Fluid Dynamics simulations [112].

10.2 Future Research Perspectives: Towards the Development of a Digital Twin

One of the main limitations currently encountered in the management of vascular diseases is the lack of effective strategies for prevention and clinical follow-up. Although the study of hemodynamics and the identification of critical parameters play a central role in the initial evaluation of the patient, there is a clear shortage of tools and methodologies capable of predicting the temporal progression of the disease. This gap results in a concrete difficulty in preventing clinical deterioration or the onset of more severe pathological conditions, with potentially fatal consequences. This highlights the need to develop diagnostic approaches and monitoring systems that are continuous, dynamic, and personalized, capable of providing an accurate and constantly updated representation of the patient's clinical condition. In this context, the paradigm of personalized medicine becomes particularly relevant. This model aims to integrate biological, physiological, environmental, and behavioral data specific to each individual in order to develop tailored therapeutic strategies [113]. A concrete example of the application of this concept is the Digital Twin (DT), a system that is proving to be crucial for the management and continuous monitoring of health. The Digital Twin is a digital and virtual representation of a physical entity, in this case, the patient, that allows real-time simulation of their health status [114]. There is a bidirectional correspondence between the real patient and their digital twin: on one hand, the patient's clinical data are continuously used to update the digital model; on the other hand, the virtual twin reflects the patient's physical condition and enables simulations and forecasts regarding their clinical status. This approach allows for the generation of highly realistic virtual data, comparable to those observed in clinical practice. [115].

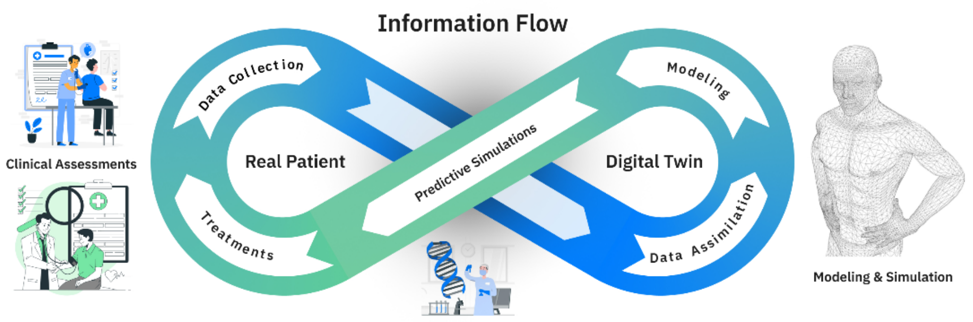


Figure 10.1: Information flow of a Digital Twin with a bidirectional feedback loop [116]

To develop a DT system, it is essential to have a model capable of assimilating data from the real patient and integrating them into the digital simulation in real time. A promising strategy to achieve this goal is represented by PINNs. PINNs, which incorporate the physical laws underlying biological processes, allow for the effective integration of clinical data with mathematical models, even in the presence of limited or noisy datasets. The integration of deep learning tools with data assimilation techniques therefore enables the continuous updating of the digital twin's state, making it possible to generate personalized predictions about the clinical evolution of the real patient and to support medical decision-making.

Appendix

Introduction to the Algorithm with the Python Library DeepXDE

The implementation of codes for solving partial differential equations using PINNs has been carried out with the Python library DeepXDE, developed by Lu et al. [103]. This library enables writing compact yet highly versatile code that closely follows the underlying mathematical formulations. In particular, DeepXDE can handle both forward problems, given initial and boundary conditions, and inverse problems, provided additional data are available.
Solving differential equations in DeepXDE primarily involves defining the problem using built-in modules, which include: the computational domain, PDE equations, boundary and initial conditions, constraints, training data, neural network architecture, and training hyperparameters.

Step-by-Step Guide to Solve PDEs with DeepXDE

To demonstrate the procedure, we focus on solving the steady Navier-Stokes equations within a square domain. The following provides a step-by-step explanation of how to define and solve this problem using DeepXDE.

1. **Definition of the computational domain**
    - The spatial domain of the problem is defined using the `geom` module. This domain may be a simple interval, a 2D or 3D shape, a more complex region, or a geometry defined by a point set, depending on the problem requirements.
    - The temporal domain can be defined separately if the problem is time-dependent.
2. **Specification of the PDE**
    - The PDE is expressed using TensorFlow syntax.
    - This involves translating the mathematical expressions, including derivatives and source terms, into code.
3. **Setting boundary and initial conditions**

- Boundary conditions (Dirichlet, Neumann, or Robin) that the solution must satisfy are specified.
- For time-dependent problems, initial conditions describing the system at the initial time are also specified.

4. **Preparation of the training data**
   - The geometry, PDE, and boundary/initial conditions are combined into a dataset using `data.PDE` (for steady-state problems) or `data.TimePDE` (for time-dependent problems).
   - Training points can be specified in two ways:
     - Specific point locations can be manually defined.
     - The number of points can be specified, allowing DeepXDE to automatically sample them on a grid or randomly.
5. **Construction of the neural network**
   - A deep neural network architecture is designed using the `nn` module.
   - The network consists of input, hidden, and output layers, along with activation functions (e.g., hyperbolic tangent, logistic sigmoid, or rectified linear unit) that approximate the PDE solution.
6. **Definition of the model**
   - A model instance is created by integrating the PDE dataset (Step 4) with the neural network architecture (Step 5).
   - This establishes the PINN framework, where the network is trained to satisfy both the differential equation and the boundary conditions.
7. **Compilation of the model**
   - The model is compiled using `model.compile` to set optimization hyperparameters, such as:
     - The optimizer (e.g., Adam, L-BFGS).
     - The learning rate, which affects training stability and convergence.
     - Loss weights, which balance the contributions of different terms (e.g., PDE residuals, boundary constraints) in the training objective.
8. **Training the model**
   - Training is initiated using `model.train` starting from:
     - Random initialization (default behavior).
     - A pre-trained model, restored with the `model.restore` command to continue training.

- Training progress can be monitored and controlled through callbacks, which allow for dynamic learning rate scheduling, early stopping, and logging of performance metrics.

9. **Prediction**

   - After training completion, `model.predict` is utilized to evaluate the PDE solution at desired locations.

By following this structured workflow, DeepXDE simplifies the implementation of PINNs, making it a powerful and user-friendly tool for solving both forward and inverse PDE problems.

Bibliography

[1] George Keith Batchelor. *An introduction to fluid dynamics.* Cambridge university press, 2000.

[2] David N Ku and Don P Giddens. “Pulsatile flow in a model carotid bifurcation.” In: *Arteriosclerosis: An Official Journal of the American Heart Association, Inc.* 3.1 (1983), pp. 31–39.

[3] Arnav Kumar et al. “Low coronary wall shear stress is associated with severe endothelial dysfunction in patients with nonobstructive coronary artery disease”. In: *JACC: Cardiovascular Interventions* 11.20 (2018), pp. 2072–2080.

[4] Xuelan Zhang et al. “Physics-informed neural networks (PINNs) for 4D hemodynamics prediction: An investigation of optimal framework based on vascular morphology”. In: *Computers in Biology and Medicine* 164 (2023), p. 107287.

[5] Joel H Ferziger, Milovan Perić, and Robert L Street. *Computational methods for fluid dynamics.* springer, 2019.

[6] Ram Kumar Raman, Yogesh Dewang, and Jitendra Raghuwanshi. “A review on applications of computational fluid dynamics”. In: *Int. j. LNCT* 2.6 (2018), pp. 137–143.

[7] Philippe R Spalart and V Venkatakrishnan. “On the role and challenges of CFD in the aerospace industry”. In: *The Aeronautical Journal* 120.1223 (2016), pp. 209–232.

[8] Elijah Hao Wei Ang, Guangjian Wang, and Bing Feng Ng. “Physics-informed neural networks for low Reynolds number flows over cylinder”. In: *Energies* 16.12 (2023), p. 4558.

[9] Jan Nordstrom. “Well posed problems and boundary conditions in computational fluid dynamics”. In: *22nd AIAA Computational Fluid Dynamics Conference.* 2015, p. 3197.

[10] Amirhossein Arzani, Jian-Xun Wang, and Roshan M D’Souza. “Uncovering near-wall blood flow from sparse data with physics-informed neural networks”. In: *Physics of Fluids* 33.7 (2021).

[11] Francesco Ballarin et al. “Fast simulations of patient-specific haemodynamics of coronary artery bypass grafts based on a POD–Galerkin method and a vascular shape parametrization”. In: *Journal of Computational Physics* 315 (2016), pp. 609–628.

[12] Geir Evensen. *Data Assimilation: The Ensemble Kalman Filter*. Springer Science & Business Media, 2009.

[13] Caterina Buizza et al. “Data learning: Integrating data assimilation and machine learning”. In: *Journal of Computational Science* 58 (2022), p. 101525.

[14] M. Asch, M. Bocquet, and M. Nodet. *Data Assimilation: Methods, Algorithms, and Applications*. SIAM, 2016.

[15] Francesco Regazzoni, Dominique Chapelle, and Philippe Moireau. “Combining data assimilation and machine learning to build data-driven models for unknown long time dynamics—applications in cardiovascular modeling”. In: *International Journal for Numerical Methods in Biomedical Engineering* 37.7 (2021), e3471.

[16] Amirhossein Arzani, Jian-Xun Wang, and Roshan M D’Souza. “Uncovering near-wall blood flow from sparse data with physics-informed neural networks”. In: *Physics of Fluids* 33.7 (2021).

[17] Peter Benner, Mario Ohlberger, Albert Cohen, and Karen Willcox. *Model reduction and approximation: theory and algorithms*. SIAM, 2017.

[18] Jan S Hesthaven, Gianluigi Rozza, Benjamin Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*. Vol. 590. Springer, 2016.

[19] Guglielmo Padula, Michele Girfoglio, and Gianlugi Rozza. “A brief review of reduced order models using intrusive and non-intrusive techniques”. In: *PAMM* 24.4 (2024), e202400210.

[20] Philip Heger, Daniel Hilger, Markus Full, and Norbert Hosters. “Investigation of physics-informed deep learning for the prediction of parametric, three-dimensional flow based on boundary data”. In: *Computers & Fluids* 278 (2024), p. 106302.

[21] Gianluigi Rozza et al. “Advances in reduced order methods for parametric industrial problems in computational fluid dynamics”. In: *arXiv preprint arXiv:1811.08319* (2018).

[22] George Em Karniadakis et al. “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[23] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[24] David J Livingstone. *Artificial neural networks: methods and applications*. Vol. 458. Springer, 2008.

[25] Andreas C Neves, Ignacio González, John Leander, and Raid Karoumi. “A new approach to damage detection in bridges using machine learning”. In: *International Conference on Experimental Vibration Analysis for Civil Engineering Structures*. Springer. 2017, pp. 73–84.

[26] Dario Floreano and Claudio Mattiussi. “Manuale sulle reti neurali”. In: (2002).

[27] AD Dongare, RR Kharde, Amit D Kachare, et al. “Introduction to artificial neural network”. In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012), pp. 189–194.

[28] Evelyn Herberg. “Lecture Notes: Neural Network Architectures”. In: *arXiv preprint arXiv:2304.05133* (2023).

[29] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. “Activation functions: Comparison of trends in practice and research for deep learning”. In: *arXiv preprint arXiv:1811.03378* (2018).

[30] Johannes Lederer. “Activation functions in artificial neural networks: A systematic overview”. In: *arXiv preprint arXiv:2101.09957* (2021).

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[32] Robert Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *Neural networks for perception.* Elsevier, 1992, pp. 65–93.

[33] Matías Roodschild, Jorge Gotay Sardiñas, and Adrián Will. “A new approach for the vanishing gradient problem on sigmoid activation”. In: *Progress in Artificial Intelligence* 9.4 (2020), pp. 351–360.

[34] *Sigmoid funtion.* URL: `https://www.sciencedirect.com/topics/computer-science/sigmoid-function`.

[35] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. “Dying relu and initialization: Theory and numerical examples”. In: *arXiv preprint arXiv:1903.06733* (2019).

[36] Juan Terven, Diana M Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A Chavez-Urbiola, and Julio A Romero-Gonzalez. “Loss functions and metrics in deep learning”. In: *arXiv preprint arXiv:2307.02694* (2023).

[37] Aryan Jadon, Avinash Patil, and Shruti Jadon. “A comprehensive survey of regression-based loss functions for time series forecasting”. In: *International Conference on Data Management, Analytics & Innovation.* Springer. 2024, pp. 117–147.

[38] Juan Terven, Diana M Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A Chavez-Urbiola, and Julio A Romero-Gonzalez. “Loss functions and metrics in deep learning”. In: *arXiv preprint arXiv:2307.02694* (2023).

[39] Alfio Quarteroni, Paola Gervasio, and Francesco Regazzoni. “Combining physics-based and data-driven models: advancing the frontiers of research with Scientific Machine Learning”. In: *arXiv preprint arXiv:2501.18708* (2025).

[40] Grégoire Montavon, Geneviève Orr, and Klaus-Robert Müller. *Neural networks: tricks of the trade.* Vol. 7700. springer, 2012.

[41] Yanzhao Wu et al. "Demystifying learning rate policies for high accuracy training of deep neural networks". In: *2019 IEEE International conference on big data (Big Data)*. IEEE. 2019, pp. 1971–1980.

[42] In: ().

[43] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[44] URL: `https://medium.com/data-science/gradient-descent-algorithm-a-deep-dive-cf04e8115f21`.

[45] James Gabbard and Daniel Miller. "Machine Learning from Scratch: Stochastic Gradient Descent and Adam Optimizer". In: (2018).

[46] URL: `https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/`.

[47] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[48] Yanli Liu, Yuan Gao, and Wotao Yin. "An improved analysis of stochastic gradient descent with momentum". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18261–18271.

[49] Xiaoyu Li, Mingrui Liu, and Francesco Orabona. "On the last iterate convergence of momentum methods". In: *International Conference on Algorithmic Learning Theory*. PMLR. 2022, pp. 699–717.

[50] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012), p. 2.

[51] Vitaly Bushaev. "Understanding RMSprop-faster neural network learning". In: *Towards Data Science* (2018).

[52] Tijmen Tieleman. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), p. 26.

[53] Oles Hospodarskyy, Vasyl Martsenyuk, Nataliia Kukharska, Andriy Hospodarskyy, and Sofiia Sverstiuk. "Understanding the Adam Optimization Algorithm in Machine Learning". In: (2024).

[54] Diederik P Kingma. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[55] C. G. Broyden, R. Fletcher, D. Goldfarb, and D. F. Shanno. "A class of methods for solving nonlinear simultaneous equations". In: *Mathematics of Computation* 24.111 (1970), pp. 221–231.

[56] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[57] *BFGS in a Nutshell: An Introduction to Quasi-Newton Methods*. 2020. URL: `\url{https://towardsdatascience.com/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504/}`.

[58] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[59] Salvatore Cuomo et al. "Scientific machine learning through physics–informed neural networks: Where we are and what's next". In: *Journal of Scientific Computing* 92.3 (2022), p. 88.

[60] Tarik Sahin, Max von Danwitz, and Alexander Popp. "Solving forward and inverse problems of contact mechanics using physics-informed neural networks". In: *Advanced Modeling and Simulation in Engineering Sciences* 11.1 (2024), p. 11.

[61] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18.153 (2018), pp. 1–43.

[62] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[63] Charles C Margossian. "A review of automatic differentiation and its efficient implementation". In: *Wiley interdisciplinary reviews: data mining and knowledge discovery* 9.4 (2019), e1305.

[64] Robert Edwin Wengert. "A simple automatic derivative evaluation program". In: *Communications of the ACM* 7.8 (1964), pp. 463–464.

[65] Barak A Pearlmutter and Jeffrey Mark Siskind. "Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 30.2 (2008), pp. 1–36.

[66] *What's Automatic Differentiation?* URL: `https://huggingface.co/blog/andmholm/what-is-automatic-differentiation`.

[67] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 426 (2021), p. 109951.

[68] Alfio Quarteroni. *Modellistica numerica per problemi differenziali.* Springer Science & Business Media, 2013.

[69] Pavel B Bochev and Max D Gunzburger. *Least-squares finite element methods.* Vol. 166. Springer Science & Business Media, 2009.

[70] Chengping Rao, Hao Sun, and Yang Liu. "Physics-informed deep learning for incompressible laminar flows". In: *Theoretical and Applied Mechanics Letters* 10.3 (2020), pp. 207–212.

[71] Natarajan Sukumar and Ankit Srivastava. "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 389 (2022), p. 114333.

[72] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. “Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data”. In: *Computer Methods in Applied Mechanics and Engineering* 361 (2020), p. 112732.

[73] Lu Lu et al. “Physics-informed neural networks with hard constraints for inverse design”. In: *SIAM Journal on Scientific Computing* 43.6 (2021), B1105–B1132.

[74] John G. Heywood, Rolf Rannacher, and Stefan Turek. “Artificial boundaries and flux and pressure conditions for the incompressible Navier–Stokes equations”. In: *International Journal for Numerical Methods in Fluids* 22.5 (1996), pp. 325–352.

[75] Frank Gijsen et al. “Expert recommendations on the assessment of wall shear stress in human coronary arteries: existing methodologies, technical considerations, and clinical applications”. In: *European heart journal* 40.41 (2019), pp. 3421–3433.

[76] Valentina Mazzi et al. “Early atherosclerotic changes in coronary arteries are associated with endothelium shear stress contraction/expansion variability”. In: *Annals of Biomedical Engineering* 49 (2021), pp. 2606–2621.

[77] Kristopher S Cunningham and Avrum I Gotlieb. “The role of shear stress in the pathogenesis of atherosclerosis”. In: *Laboratory investigation* 85.1 (2005), pp. 9–23.

[78] Avedis Assadour Ekmejian et al. “Advances in the computational assessment of disturbed coronary flow and wall shear stress: a contemporary review”. In: *Journal of the American Heart Association* 13.19 (2024), e037129.

[79] Saeyoung Kim et al. “Dynamic coronary blood flow velocity and wall shear stress estimation using ultrasound in an ex vivo porcine heart”. In: *Cardiovascular engineering and technology* 15.1 (2024), pp. 65–76.

[80] DP Giddens, CK Zarins, and S Glagov. “The role of fluid mechanics in the localization and detection of atherosclerosis”. In: (1993).

[81] Peter Libby, Paul M Ridker, and Göran K Hansson. “Progress and challenges in translating the biology of atherosclerosis”. In: *Nature* 473.7347 (2011), pp. 317–325.

[82] Ricvan Dana Nindrea and Asni Hasanuddin. “Non-modifiable and modifiable factors contributing to recurrent stroke: A systematic review and meta-analysis”. In: *Clinical Epidemiology and Global Health* 20 (2023), p. 101240.

[83] Elena Bacigalupi et al. “Biomechanical factors and atherosclerosis localization: insights and clinical applications”. In: *Frontiers in Cardiovascular Medicine* 11 (2024), p. 1392702.

[84] *stent e neoaterosclerosi a che punto siamo*. URL: https://www.centrolottainfarto.com/allnews/stent-e-neoaterosclerosi-a-che-punto-siamo/.

[85] Imran Shah et al. "Impact of the stent footprint on endothelial wall shear stress in patient-specific coronary arteries: A computational analysis from the SHEAR-STENT trial". In: *Computer Methods and Programs in Biomedicine* 266 (2025), p. 108762.

[86] Konstantinos C Koskinas, Yiannis S Chatzizisis, Antonios P Antoniadis, and George D Giannoglou. "Role of endothelial shear stress in stent restenosis and thrombosis: pathophysiologic mechanisms and implications for clinical translation". In: *Journal of the American College of Cardiology* 59.15 (2012), pp. 1337–1349.

[87] Claudio Chiastra et al. "Healthy and diseased coronary bifurcation geometries influence near-wall and intravascular flow: A computational exploration of the hemodynamic risk". In: *Journal of biomechanics* 58 (2017), pp. 79–88.

[88] N Pinho et al. "Correlation between geometric parameters of the left coronary artery and hemodynamic descriptors of atherosclerosis: FSI and statistical study". In: *Medical & biological engineering & computing* 57 (2019), pp. 715–729.

[89] WR Dean. "The stream-line motion of fluid in a curved pipe". In: *Phil. Mag.* 5 (1928), pp. 673–695.

[90] Natalya Vorobtsova et al. "Effects of vessel tortuosity on coronary hemodynamics: an idealized and patient-specific computational study". In: *Annals of biomedical engineering* 44 (2016), pp. 2228–2239.

[91] Abdulrajak Buradi and Arun Mahalingam. "Impact of coronary tortuosity on the artery hemodynamics". In: *Biocybernetics and Biomedical Engineering* 40.1 (2020), pp. 126–147.

[92] Jianfei Song, Smaïne Kouidri, and Farid Bakir. "Numerical study on flow topology and hemodynamics in tortuous coronary artery with symmetrical and asymmetrical stenosis". In: *Biocybernetics and Biomedical Engineering* 41.1 (2021), pp. 142–155.

[93] Lucas H Timmins et al. "Oscillatory wall shear stress is a dominant flow characteristic affecting lesion progression patterns and plaque vulnerability in patients with coronary artery disease". In: *Journal of The Royal Society Interface* 14.127 (2017), p. 20160972.

[94] Paul D Morris, Rasha Kadem Al-Lamee, and Colin Berry. "Coronary physiological assessment in the catheter laboratory: haemodynamics, clinical assessment and future perspectives". In: *Heart* 108.21 (2022), pp. 1737–1746.

[95] Biyue Liu et al. "The wall shear stress of a pulsatile blood flow in a patient specific stenotic right coronary artery". In: *Engineering* 5.10 (2013), pp. 396–399.

[96] Fadi J Sawaya et al. "Contemporary approach to coronary bifurcation lesion treatment". In: *JACC: Cardiovascular Interventions* 9.18 (2016), pp. 1861–1878.

[97] Lorenzo Genuardi et al. “Local fluid dynamics in patients with bifurcated coronary lesions undergoing percutaneous coronary interventions”. In: *Cardiology Journal* 28.2 (2021), pp. 321–329.

[98] C Shen et al. “Secondary flow in bifurcations–important effects of curvature, bifurcation angle and stents”. In: *Journal of biomechanics* 129 (2021), p. 110755.

[99] Alessandro Candreva et al. “Risk of myocardial infarction based on endothelial shear stress analysis using coronary angiography”. In: *Atherosclerosis* 342 (2022), pp. 28–35.

[100] Frank Gijsen et al. “Expert recommendations on the assessment of wall shear stress in human coronary arteries: existing methodologies, technical considerations, and clinical applications”. In: *European heart journal* 40.41 (2019), pp. 3421–3433.

[101] John R Womersley. “Method for the calculation of velocity, rate of flow and viscous drag in arteries when the pressure gradient is known”. In: *The Journal of physiology* 127.3 (1955), p. 553.

[102] Alfio Quarteroni, Fausto Saleri, and Alessandro Veneziani. “Factorization methods for the numerical approximation of Navier–Stokes equations”. In: *Computer methods in applied mechanics and engineering* 188.1-3 (2000), pp. 505–526.

[103] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. “DeepXDE: A deep learning library for solving differential equations”. In: *SIAM review* 63.1 (2021), pp. 208–228.

[104] John Kim and Parviz Moin. “Application of a fractional-step method to incompressible Navier-Stokes equations”. In: *Journal of computational physics* 59.2 (1985), pp. 308–323.

[105] Salvatore P Sutera and Richard Skalak. “The history of Poiseuille’s law”. In: *Annual review of fluid mechanics* 25.1 (1993), pp. 1–20.

[106] J Theodore Dodge Jr, B Greg Brown, Edward L Bolson, and Harold T Dodge. “Lumen diameter of normal human coronary arteries. Influence of age, sex, anatomic variation, and left ventricular hypertrophy or dilation.” In: *Circulation* 86.1 (1992), pp. 232–246.

[107] William Reginald Dean. “Fluid motion in a curved channel”. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 121.787 (1928), pp. 402–420.

[108] Phillip M Ligrani. “A study of Dean vortex development and structure in a curved rectangular channel with aspect ratio of 40 at Dean numbers up to 430”. In: (1994).

[109] Aland Santamarina, Erlend Weydahl, John M Siegel, and James E Moore. “Computational analysis of flow in a curved tube model of the coronary arteries: effects of time-varying curvature”. In: *Annals of biomedical engineering* 26 (1998), pp. 944–954.

[110] Michael Markl, Alex Frydrychowicz, Sebastian Kozerke, Mike Hope, and Oliver Wieben. “4D flow MRI”. In: *Journal of Magnetic Resonance Imaging* 36.5 (2012), pp. 1015–1036.

[111] Christian Poelma. “Ultrasound imaging velocimetry: a review”. In: *Experiments in Fluids* 58 (2017), pp. 1–28.

[112] Georgios Kissas et al. “Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 358 (2020), p. 112623.

[113] V. Bollati, L. Ferrari, V. Leso, and I. Iavicoli. “Personalised Medicine: implication and perspectives in the field of occupational health”. In: *Medicina del Lavoro* 111.6 (2020), pp. 425–444.

[114] Evangelia Katsoulakis et al. “Digital twins for health: a scoping review”. In: *npj Digital Medicine* 7.77 (2024).

[115] Kang Zhang et al. “Concepts and applications of digital twins in healthcare and medicine”. In: *Review* 5.8 (2024).

[116] Alessandro Veneziani, Annalisa Quaini, Marco Tezzele, Omer San, and Traian Iliescu. “Digital Twins in Coronary Artery Disease: A Mathematical Roadmap”. In: (2025). Submitted.