جامعة خليفة
Khalifa University

A Study of Physics-Informed Neural Networks for Incompressible Flow: Benchmark Simulations

Mustafa Ali

Student ID: 100059544

Senior Research Project

Department of Mathematics

Supervisor: Dr. Aymen Laadhari, Khalifa University
Co-Supervisor: Dr. Mohamed Soufiane Jouini, Khalifa University

April 24, 2026

Abstract

Mustafa Ali, "**A Study of Physics-Informed Neural Networks for Incompressible Flow: Benchmark Simulations**", Senior Research Project, Department of Mathematics, Khalifa University, United Arab Emirates, April 24, 2026.

This project investigates the application of Physics-Informed Neural Networks (PINNs) to the numerical solution of incompressible fluid flow problems. PINNs provide a mesh-free alternative to classical computational fluid dynamics methods by embedding the governing partial differential equations directly into the training process of a neural network.

The first chapter introduces the mathematical formulation of a multiphysics problem involving biomembranes, with emphasis on level set methods, geometric quantities, and the Helfrich bending energy. The associated fourth-order derivatives highlight the major computational challenges of such problems.

The second chapter presents the fundamental concepts of neural networks and their extension to physics-informed learning. The PINN framework is then applied to the incompressible Navier–Stokes equations using a mixed-variable formulation that improves stability and reduces derivative order in the loss function.

Finally, the methodology is validated through some benchmark simulations of steady and transient flow in a circular cylinder. The results show good qualitative agreement with reference solutions from the literature, while also highlighting some limitations mainly in the computational cost.

Although the full coupling between fluid flow and membrane dynamics is not implemented, this work establishes preliminary tools for future work on applying PINNs to complex multiphysics problems involving biomembranes.

Contents

Abstract **iii**

Contents . . . vi

1 Fluid–Membrane Modeling: Basics of a Challenging Problem **1**

1.1 Introduction . . . 1
1.2 Mathematical Formulation of the Interface . . . 1
1.3 Level Set Method . . . 2
1.4 Differential Geometry of the Interface . . . 3
1.5 Verification with a Fully Worked Example . . . 4
1.6 Physics of Fluid Membranes . . . 5
1.7 The Computational Challenge . . . 7

2 Neural Networks for PDE Approximation **9**

2.1 Function Approximation . . . 9
2.2 The Multilayer Perceptron . . . 9
2.3 Activation Functions . . . 10
2.4 The Minimization Process . . . 10
2.5 Theoretical Justification . . . 11
2.6 The Physics-Informed Loss Functional . . . 11
2.7 Optimization Algorithms . . . 12
2.8 Automatic Differentiation . . . 13

3 PINNs for Incompressible Flow: Formulation and Benchmark Simulations **14**

3.1 Introduction . . . 14
3.2 Navier-Stokes Equations . . . 15
3.3 Mixed-Variable Formulation . . . 15
3.4 Residuals . . . 16
3.5 Boundary Conditions as Data . . . 16
3.6 The Composite Loss Function . . . 17
3.7 Numerical Validation . . . 17
3.8 Reproduction of the Steady Benchmark . . . 18
3.9 Steady-State Results . . . 19
3.10 Convergence Comparison . . . 22

3.11 Reproduction of the Transient Benchmark 22
3.12 Transient Results 23
3.13 Comparison with Traditional Methods 28
3.14 Continuous Time Modeling 28

Conclusion **30**

Bibliography **31**

Chapter 1

Fluid–Membrane Modeling: Basics of a Challenging Problem

1.1 Introduction

Simulating vesicles, such as red blood cells or lipid bilayers suspended in a viscous fluid, is a specific type of mathematical challenge known as a Free Boundary Problem. In a standard Partial Differential Equation (PDE) problem, like calculating fluid flow inside a stationary pipe, the shape of the domain is fixed and we know it from the start.

The vesicle problem is different because the boundary of the domain, denoted as $\Gamma(t)$, is not fixed. First, the position of this boundary is unknown and finding it is actually part of the solution we are trying to calculate. Second, it is dynamic, meaning it moves and changes shape over time based on the speed of the fluid around it. Finally, the boundary is active. It does not just move passively; it exerts forces like bending and tension back onto the fluid. This creates a coupled system where the fluid moves the membrane, and the membrane pushes back on the fluid [2].

1.2 Mathematical Formulation of the Interface

To track this moving boundary $\Gamma(t)$, mathematicians generally use one of two frameworks. It is important to understand the difference between them to see why we chose the Level Set Method for this project.

The first approach is the Lagrangian description. This is the most intuitive way to think about the problem. We explicitly mark the interface using a mesh of connected points or nodes. If we look at a single particle at a position X on the surface, it moves according to the trajectory equation:

$$\frac{dX}{dt} = u(X, t) \tag{1.1}$$

where u is the fluid velocity. While this makes sense physically, it has a major drawback when we try to simulate it. As the vesicle deforms, especially in strong flows, the triangles in the mesh

can get twisted, stretched, or even overlap. To fix this, the computer has to constantly stop and remesh the surface to keep the calculation stable, which takes a lot of computing power.

The second approach is the Eulerian description. In this method, we do not track the particles on the surface directly. Instead, we use a fixed grid that covers the entire space. We define a scalar function, called $\phi(x,t)$, over this whole domain. The interface is then defined implicitly [19]. It is simply the specific line or contour where this function equals zero. This method handles large changes in shape naturally and does not suffer from the mesh distortion problems found in the Lagrangian approach. For this research, we adopt the Eulerian framework, specifically the Level Set Method [8, 6].

1.3 Level Set Method

We define our computational domain Ω in two or three dimensions ($\mathbb{R}^d$). The interface $\Gamma(t)$ is represented as the zero-level set of a continuous scalar function ϕ. This function allows us to partition the entire domain based on the sign of ϕ.

We define the inside of the vesicle (the intracellular fluid) as the region where the function is negative. The membrane itself is the set of points where the function is exactly zero. The outside of the vesicle (the extracellular fluid) is the region where the function is positive. Mathematically, we write this partition as:

$$\begin{cases} \phi(x,t) < 0 & \text{if } x \in \Omega^- \text{ (Inside)} \\ \phi(x,t) = 0 & \text{if } x \in \Gamma(t) \text{ (Membrane)} \\ \phi(x,t) > 0 & \text{if } x \in \Omega^+ \text{ (Outside)} \end{cases} \tag{1.2}$$

However, we cannot just use any function for this. If the function becomes too flat (where the slope is near zero) or too steep near the interface, our calculations for curvature will become unstable. To prevent this, we impose a strict condition that ϕ must be a Signed Distance Function (SDF). This means that for any point x in the domain, the value of $\phi(x)$ must be equal to the shortest distance from that point to the interface. In terms of calculus, this requires the function to satisfy the Eikonal Equation:

$$|\nabla \phi(x)| = 1, \quad \forall x \in \Omega \tag{1.3}$$

This condition ensures that the magnitude of the gradient is always equal to 1. This normalization is critical because it simplifies the complex formulas we will need later to calculate high-order geometric derivatives, such as the 4th-order derivatives required for the bending energy [13].

Using our unit circle $\phi(x,y) = \sqrt{x^2+y^2} - 1$, we can explicitly define these regions. Evaluating the function at the origin $(0,0)$ yields $\phi = -1$, placing it strictly inside the intracellular fluid (Ω^-). Evaluating it at $(2,0)$ yields $\phi = 1$, placing it in the extracellular fluid (Ω^+). A point at $(1,0)$ yields $\phi = 0$, placing it exactly on the active membrane interface ($\Gamma(t)$).

1.4 Differential Geometry of the Interface

In the Lagrangian framework (using a mesh), finding geometric properties like the normal vector is easy because we just look at the orientation of the flat triangles. In our Eulerian framework, we do not have triangles. Instead, we must calculate these properties directly from the derivatives of our level set function ϕ.

Unit Normal Vector

The normal vector n represents the direction perpendicular to the surface at any given point. Since our surface is defined by a constant value of ϕ (specifically zero), the gradient vector $\nabla\phi$ will always point perpendicular to the interface.

To get a unit normal vector (a vector with a length of 1), we simply take the gradient and divide it by its own magnitude:

$$n = \frac{\nabla\phi}{|\nabla\phi|} \tag{1.4}$$

In three dimensions, the gradient is simply the vector of partial derivatives:

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right)^T \tag{1.5}$$

If we successfully maintain the Signed Distance Function property where $|\nabla\phi| = 1$, this simplifies nicely. The denominator becomes 1, and the normal vector is just the gradient itself.

Mean Curvature

Curvature is a measure of how much the surface is bending. Mathematically, it measures how the normal vector changes as we move along the surface. We define the mean curvature H as the negative divergence of the normal vector field:

$$H = -\nabla \cdot n = -\nabla \cdot \left(\frac{\nabla\phi}{|\nabla\phi|}\right) \tag{1.6}$$

There is a simple physical interpretation for this. If the divergence is positive, the normal vectors are spreading out, which means the surface is convex (like a ball). If the divergence is negative, the vectors are coming together, meaning the surface is concave.

In terms of derivatives, if our function is a perfect signed distance function, the curvature is simply the Laplacian (the sum of second derivatives) of the level set function:

$$H = -\Delta\phi = -\left(\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2}\right) \tag{1.7}$$

This equation is very important because it shows that curvature is a second-order geometric quantity.

1.5 Verification with a Fully Worked Example

To prove that these definitions are mathematically consistent, we will perform a full manual calculation for a unit circle centered at the origin.

Level Set Definition

We already defined the signed distance function for a circle of radius $R = 1$:

$$\phi(x, y) = \sqrt{x^2 + y^2} - 1 \tag{1.8}$$

Let $r = \sqrt{x^2 + y^2}$. Thus, the function can be written as $\phi(x, y) = r - 1$.

Gradient and Normal Calculation

To find the unit normal vector n, we first compute the gradient $\nabla\phi$. Applying the chain rule to differentiate with respect to x:

$$\frac{\partial \phi}{\partial x} = \frac{\partial}{\partial x}\left((x^2 + y^2)^{1/2} - 1\right) = \frac{1}{2}(x^2 + y^2)^{-1/2}(2x) = \frac{x}{\sqrt{x^2 + y^2}} = \frac{x}{r} \tag{1.9}$$

By symmetry, the partial derivative with respect to y is:

$$\frac{\partial \phi}{\partial y} = \frac{y}{\sqrt{x^2 + y^2}} = \frac{y}{r} \tag{1.10}$$

The gradient vector is therefore $\nabla\phi = \left(\frac{x}{r}, \frac{y}{r}\right)^T$. We check its magnitude to verify if the Eikonal equation ($|\nabla\phi| = 1$) holds:

$$|\nabla\phi| = \sqrt{\left(\frac{x}{r}\right)^2 + \left(\frac{y}{r}\right)^2} = \sqrt{\frac{x^2 + y^2}{r^2}} = \sqrt{\frac{r^2}{r^2}} = 1 \tag{1.11}$$

Since the magnitude is exactly 1, the gradient is already a unit vector. The normal vector is $n = \nabla\phi = \left(\frac{x}{r}, \frac{y}{r}\right)^T$ [10].

Mean Curvature Calculation

Next, we calculate the mean curvature $H = -\nabla \cdot n = -\left(\frac{\partial n_x}{\partial x} + \frac{\partial n_y}{\partial y}\right)$. We differentiate the x-component of the normal ($n_x = x/r$) using the quotient rule, noting that $\frac{\partial r}{\partial x} = \frac{x}{r}$:

$$\frac{\partial n_x}{\partial x} = \frac{\partial}{\partial x}\left(\frac{x}{r}\right) = \frac{1 \cdot r - x \cdot \frac{\partial r}{\partial x}}{r^2} = \frac{r - x\left(\frac{x}{r}\right)}{r^2} = \frac{r^2 - x^2}{r^3} \tag{1.12}$$

Since $r^2 = x^2 + y^2$, we substitute $r^2 - x^2 = y^2$:

$$\frac{\partial n_x}{\partial x} = \frac{y^2}{r^3} \tag{1.13}$$

By symmetry, differentiating the y-component ($n_y = y/r$) with respect to y yields:

$$\frac{\partial n_y}{\partial y} = \frac{x^2}{r^3} \tag{1.14}$$

Summing these components gives the divergence of the normal vector:

$$\nabla \cdot n = \frac{y^2}{r^3} + \frac{x^2}{r^3} = \frac{x^2 + y^2}{r^3} = \frac{r^2}{r^3} = \frac{1}{r} \tag{1.15}$$

The mean curvature is the negative divergence:

$$H = -\frac{1}{r} \tag{1.16}$$

At the interface (the membrane), $\phi = 0$, which implies $r = 1$. Evaluating the curvature at the interface gives $H = -1$. This matches the classical geometric definition of curvature for a unit circle, verifying the level set mechanics.

1.6 Physics of Fluid Membranes

Surface Tension Problem

In simple fluid interfaces, such as the surface of a water droplet, the physical behavior is primarily governed by surface tension. The interface naturally tends to minimize its total surface area A to reach an equilibrium state. We can describe this mathematically using an energy functional:

$$E_{tension} = \sigma \int_{\Gamma} dA \tag{1.17}$$

where σ is the constant surface tension coefficient. When we minimize this energy, we find that a force acts normal to the surface, pulling it inward. This force creates a jump in pressure across the interface, which is described by the famous Young-Laplace equation:

$$[p] = p_{in} - p_{out} = \sigma H \tag{1.18}$$

In the context of Computational Fluid Dynamics (CFD), we do not usually apply this pressure jump directly. Instead, we implement it as a singular body force term, denoted as f_{st}, inside the Navier-Stokes equations [7, 15]. This force is localized strictly to the interface using the Dirac delta function $\delta(\phi)$:

$$f_{st}(x) = \sigma H(x) n(x) \delta(\phi) \tag{1.19}$$

Fully implicit numerical solvers have been developed for this problem; see, for instance, the finite element approach presented in [15].

Biomembranes and Helfrich Bending Energy

Vesicles, such as lipid bilayers, behave differently from simple water droplets. The lipid molecules in the membrane are packed very tightly together. This means the membrane area is effectively incompressible; it cannot stretch or shrink locally. However, the membrane is very thin and

flexible, so it offers resistance to bending.

The shape of a vesicle is determined by minimizing the Helfrich Bending Energy rather than just surface area [4]. The energy functional is defined as:

$$E_{bend} = \frac{k_B}{2}\int_\Gamma (H - c_0)^2 dA + \int_\Gamma \lambda dA \tag{1.20}$$

Here, k_B represents the bending rigidity, which tells us how stiff the membrane is. The term c_0 is the spontaneous curvature, which we assume is zero for symmetric membranes. The variable λ is a Lagrange multiplier. It acts like a local surface tension that adjusts dynamically to enforce the constraint that the local area must remain constant.

Example: Bending Energy of a Unit Circle

We can calculate the base Helfrich bending energy for our stationary unit circle. Assuming a spontaneous curvature of $c_0 = 0$ and ignoring the area constraint term ($\lambda = 0$) for this static case, the energy depends entirely on the mean curvature. As derived in Section 1.5, the curvature of the unit circle is constant everywhere at $H = -1$.

Substituting this into the energy functional:

$$E_{bend} = \frac{k_B}{2}\int_\Gamma (-1 - 0)^2 dA = \frac{k_B}{2}\int_\Gamma 1 dA \tag{1.21}$$

The integral $\int_\Gamma dA$ evaluates to the total perimeter of the unit circle, which is $2\pi(1) = 2\pi$. Therefore, the total bending energy is:

$$E_{bend} = \frac{k_B}{2}(2\pi) = \pi k_B \tag{1.22}$$

This demonstrates how the geometric variables derived from the level set function dictate the physical equilibrium states of the membrane.

The 4th-Order Variational Derivative

To find the force that the membrane exerts on the fluid, we must take the functional derivative (variation) of the Helfrich energy with respect to the surface shape. This is a complex mathematical procedure that involves the surface divergence theorem.

The resulting elastic force density is highly non-linear and involves high-order derivatives:

$$f_{mem} = \left(-k_B\left[\Delta_\Gamma H + \frac{1}{2}H(H^2 - 4K)\right] + 2H\lambda + \nabla_\Gamma \lambda \cdot n\right) n \tag{1.23}$$

In this equation, K represents the Gaussian curvature, and Δ_Γ is the Laplace-Beltrami operator, also known as the surface Laplacian. This formula is the heart of the problem because it introduces the 4th-order derivatives that make the simulation so difficult. A derivation based on shape optimization techniques is developed in [14]

1.7 The Computational Challenge

The Derivative Bottleneck

The mathematical formulation derived in the previous sections leads us to a significant hurdle. As we saw in equation 1.18, the bending force density f_{mem} depends on the surface Laplacian of the mean curvature, $\Delta_\Gamma H$.

Let us trace the order of differentiation required to compute this term from our primary variable, the level set function ϕ:

1. The normal vector n depends on the gradient $\nabla\phi$ (1st derivative).
2. The curvature H depends on the divergence of the normal $\nabla \cdot n$ (2nd derivative).
3. The surface Laplacian $\Delta_\Gamma H$ involves two more spatial derivatives of the curvature.

Consequently, the bending force depends on the fourth-order spatial derivatives of the level set function (behaves as $\nabla^4\phi$), thereby classifying the problem as a fourth-order geometric flow. While the mathematical formulation is compact, its numerical treatment is particularly delicate due to the stiffness and instabilities associated with high-order derivatives. Accurate resolution of these terms has motivated the development of advanced numerical methodologies, notably steady-state solvers [10] and implicit fully coupled time-dependent finite element approaches for membrane–fluid interaction [11, 13, 19].

The Continuity Constraint in Finite Elements

Most commercial simulation software uses the standard Finite Element Method (FEM). In FEM, we divide the domain into small elements (like triangles or tetrahedra) and approximate the solution using simple polynomial functions within each element.

Standard elements (called Lagrange elements) ensure that the solution is continuous across the boundaries between elements. This is known as C^0 continuity. However, they do not guarantee that the *slope* (derivative) is continuous; the gradient often jumps at the edges of the triangles.

For a second-order problem (such as heat transfer or standard fluid flow), C^0 continuity is sufficient. However, for a 4th-order problem, the numerical theory dictates that our basis functions must have C^1 continuity. This means both the value of ϕ and its gradient $\nabla\phi$ must be perfectly smooth across all element boundaries.

Limitations of Classical Approaches

The construction of finite element spaces with C^1 continuity is technically demanding and requires specialized elements. An alternative strategy consists in reformulating the fourth-order problem as a system of coupled second-order equations (mixed finite element method). While this approach avoids the need for C^1-continuous basis functions, it introduces additional unknowns and leads to saddle-point problems, which may suffer from stability issues and typically require carefully designed stabilization techniques and stable time-discretization techniques.

A variety of numerical methods have been developed to address these challenges; we refer the reader, for example, to [2, 18, 3, 5, 11, 1]. Further difficulties arise from mass conservation properties in numerical approximations; see, for instance, the mass correction method in [12] as well as the discussion and recent developments of robust schemes for simplier problems in [9].

These difficulties in the classical mesh-based framework are the primary motivation for exploring a mesh-free Deep Learning approach.

Chapter 2

Neural Networks for PDE Approximation

2.1 Function Approximation

In classical numerical analysis, when we want to approximate an unknown function (such as the fluid velocity $u(x)$), we typically use a combination of simple "basis functions." These are often polynomials, as used in Finite Elements, or trigonometric functions, as used in Spectral Methods.

A Deep Neural Network (DNN) offers a different approach. Instead of combining fixed polynomials, it acts as a composite, non-linear function approximation. Mathematically, we can view a neural network as a mapping $\mathcal{N}$ parameterized by a set of weights and biases θ. For our specific problem of vesicle dynamics, the input to this mapping is the coordinate vector $z = (x, y, t)$, and the output is the vector of physical fields we want to find, such as velocity and pressure $U = (u, v, p, \phi)$.

2.2 The Multilayer Perceptron

The architecture we use for Physics-Informed Neural Networks [16] is the standard Feed-Forward network, also known as the Multilayer Perceptron (MLP). This structure consists of an input layer, several "hidden" layers, and a final output layer.

We can define the mathematics of a single layer precisely [17]. If we have a network with L hidden layers, the information propagates forward through the system in a sequence of matrix operations. For any given layer l, the output $h^{(l)}$ is calculated as:

$$h^{(l)} = \sigma(W^{(l)}h^{(l-1)} + b^{(l)}) \tag{2.1}$$

In this equation, $W^{(l)}$ represents the weight matrix that connects the previous layer to the current one, and $b^{(l)}$ is the bias vector. The function $\sigma(\cdot)$ is the non-linear activation function, which is applied element-wise to every value in the vector.

2.3 Activation Functions

The activation function is what gives the neural network its power. Without it, the network would just be a series of linear matrix multiplications, which would collapse into a single linear regression regardless of how deep the network is.

For our specific problem, the choice of activation function is critical [16]. The most common activation function in modern AI is the Rectified Linear Unit (ReLU), defined as $\sigma(z) = \max(0, z)$. While ReLU works very well for image recognition, it is not suitable for solving our 4th-order differential equations. This is because the second derivative of ReLU is zero everywhere (except at the origin where it is undefined).

Since we need to compute up to the 4th derivative for the bending energy, we require a function that is smooth and infinitely differentiable. Therefore, we use the Hyperbolic Tangent (tanh) activation function:

$$\sigma(z) = \tanh(z) \tag{2.2}$$

This function is smooth, bounded, and has non-zero derivatives of all orders, making it the ideal choice for approximating the complex geometric flows of vesicles.

2.4 The Minimization Process

The "learning" in a neural network is actually an optimization problem. We aim to minimize a Loss Function $\mathcal{L}$ that measures the error between our prediction and the reality.

Gradient Calculation via Chain Rule

To minimize the loss, we must adjust the weights. We do this by calculating the gradient of the error with respect to each specific weight. This relies on the Chain Rule of calculus.

For example, if we use a Squared Error loss $\mathcal{L} = \frac{1}{2}(y_{pred} - y_{target})^2$, and we want to find how much to change the weight $w_{1,1}$, we decompose the derivative backwards from the output to the input:

$$\frac{\partial \mathcal{L}}{\partial w_{1,1}} = \frac{\partial \mathcal{L}}{\partial y_{pred}} \cdot \frac{\partial y_{pred}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_{1,1}} \tag{2.3}$$

We can solve each term analytically:

$$\frac{\partial \mathcal{L}}{\partial y_{pred}} = (y_{pred} - y_{target}) \tag{2.4}$$

$$\frac{\partial y_{pred}}{\partial h_1} = w_{2,1} \tag{2.5}$$

$$\frac{\partial h_1}{\partial w_{1,1}} = x \tag{2.6}$$

Combining these gives us the exact gradient:

$$\nabla \mathcal{L} = (y_{pred} - y_{target}) \cdot w_{2,1} \cdot x \tag{2.7}$$

This gradient tells us the direction of steepest ascent. To improve the network, we update the weight in the opposite direction using a learning rate η:

$$w_{new} = w_{old} - \eta \nabla \mathcal{L} \tag{2.8}$$

In our project, the process is identical, but the Loss Function $\mathcal{L}$ will contain the residuals of the Navier-Stokes and Helfrich equations rather than just simple data errors.

2.5 Theoretical Justification

The use of neural networks for this type of problem is supported by the Universal Approximation Theorem. In simple terms, this theorem states that a feed-forward network with even a single hidden layer can approximate any continuous function, provided it has enough neurons.

However, for our specific problem involving the sharp interface of a vesicle membrane, the function we are trying to model changes very rapidly over small distances. While a shallow network *could* theoretically work, extensions of the theorem suggest that "deep" networks (those with many layers) are far more efficient at capturing these high-frequency features. This is why we use a deep architecture rather than a wide, single-layer one.

2.6 The Physics-Informed Loss Functional

In a conventional data-driven neural network, training is simply a game of matching the network's output to a set of labeled pictures or text. In a Physics-Informed Neural Network (PINN), we change the rules. We embed the governing physical equations directly into the function we are trying to minimize [16].

The goal is to find the parameters θ that minimize a total loss functional $\mathcal{L}(\theta)$. This loss is a weighted sum of two distinct components:

$$\mathcal{L}(\theta) = w_{data}\mathcal{L}_{data} + w_{PDE}\mathcal{L}_{PDE} \tag{2.9}$$

Here, w_{data} and w_{PDE} are weights that allow us to balance the importance of matching experimental data versus satisfying the laws of physics.

The Data Loss

The first term, $\mathcal{L}_{data}$, is the standard error term used in almost all neural networks. It enforces any known boundary conditions or experimental measurements we might have. If we have a set of N_d measurements at specific locations, we calculate the Mean Squared Error (MSE) between what our network predicts and the true measured value:

$$\mathcal{L}_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} ||u_\theta(x^{(i)}) - u_{meas}^{(i)}||^2 \tag{2.10}$$

If we were running a pure simulation without any experimental data, this term would simply

enforce the boundary conditions on the walls of the domain (e.g., that the fluid velocity is zero at the walls).

The PDE Residual Loss

The second term, $\mathcal{L}_{PDE}$, is what makes the network "physics-informed." We define a residual function $r(x)$, which represents the error in the differential equation.

For example, if we look at the momentum equation for the fluid, the residual is the difference between the left-hand side (acceleration) and the right-hand side (forces). Ideally, this difference should be zero. For the Helfrich flow, we write the residual as:

$$r(x,t;\theta) = \rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) - \nabla \cdot \sigma - f_{mem}(\phi) \tag{2.11}$$

In a traditional grid-based method, we would solve this at grid nodes. In a PINN, we force this residual to be zero at a set of "Collocation Points." These are points randomly sampled scattered throughout the entire domain.

$$\mathcal{L}_{PDE} = \frac{1}{N_c}\sum_{j=1}^{N_c} ||r(x_j,t_j;\theta)||^2 \tag{2.12}$$

By minimizing this term, we compel the neural network to learn the physical laws—Mass Conservation, Momentum Balance, and Geometric Equilibrium—even in the empty spaces where we have no data.

2.7 Optimization Algorithms

The training process is essentially a navigation problem. We are trying to find the lowest point (minimum loss) in a landscape that has millions of dimensions (the parameters θ).

Gradient Descent

The most fundamental algorithm for this is Gradient Descent. The concept is simple: we calculate the gradient of the loss function, which points up the steepest hill. To minimize the loss, we take a step in the exact opposite direction:

$$\theta_{new} = \theta_{old} - \eta \nabla_\theta \mathcal{L} \tag{2.13}$$

Here, η is the learning rate. If this step size is too small, the network learns incredibly slowly. If it is too large, the optimizer might step right over the minimum and diverge, causing the simulation to crash.

The Adam Optimizer

For our specific problem—the Helfrich flow—the "loss landscape" is very difficult to navigate. It is what mathematicians call "stiff." This means it has some directions that are very steep (related to the 4th-order bending terms) and others that are very flat (related to the simple fluid flow).

Standard Gradient Descent struggles here. It tends to bounce back and forth across the steep valleys without making progress down them. To fix this, we use the Adam (Adaptive Moment Estimation) optimizer. Adam improves the process in two ways:

1. **Momentum:** It keeps a rolling average of previous gradients. This allows the optimizer to build up speed and "roll" over small bumps or local minima that would otherwise trap standard gradient descent.
2. **Adaptive Scaling:** It scales the learning rate individually for each parameter. This is crucial for us. It ensures that the weights controlling the sensitive bending forces and the weights controlling the bulk fluid flow converge at roughly the same speed.

2.8 Automatic Differentiation

A common question in this research is: "How do we actually compute a 4th derivative like $\partial^4\phi/\partial x^4$ on a computer?"

We do not use Finite Differences, which approximate the slope using two points. That method introduces truncation errors that get worse with every derivative order. We also do not use Symbolic Differentiation (like Mathematica), because the formulas would become exponentially long and slow to compute.

Instead, we use Automatic Differentiation (AD).

The Computational Graph

AD treats the neural network not as a black box, but as a "Computational Graph." Every single mathematical operation—whether it is a multiplication, a sine function, or an addition—is recorded as a node in this graph.

When we need a derivative, the computer simply applies the Chain Rule to each link in this graph, moving backwards from the output to the input.

Exactness

The most important feature of AD is that it is **exact**. It computes the derivative to machine precision. It does not estimate the slope; it calculates the analytical value of the slope based on the chain rule.

In a PINN, to compute the 4th derivative $\nabla^4\phi$, we simply tell the software to traverse this graph backwards four times. This is the technological breakthrough that makes this project feasible. It allows us to evaluate the complex, high-order Helfrich forces rigorously without ever defining a mesh or worrying about element connectivity.

Chapter 3

PINNs for Incompressible Flow: Formulation and Benchmark Simulations

3.1 Introduction

In the previous chapter, we established the mathematical architecture of Deep Neural Networks. We defined how they function as universal approximators and how Automatic Differentiation allows us to compute exact derivatives.

Now, we bridge the gap between that abstract mathematical theory and real-world physics. This chapter details the specific methodology for applying those networks to fluid dynamics, based on the foundational work of Rao et al. regarding "Physics-Informed Deep Learning for Incompressible Laminar Flows" [17].

The core idea is simple but powerful: we treat the Neural Network defined in Chapter 2 as a solution generator. We then use the physical laws of fluid motion to "grade" that solution and tell the optimizer how to improve it.

In this chapter, we first derive the mathematical formulation of the Navier-Stokes PINN. We then reproduce the benchmark flow past a cylinder from Rao et al., presenting our results directly alongside the corresponding figures from the paper to allow a clear visual comparison. It should be noted that the benchmark reproduced in this chapter follows the original Rao et al. computational geometry—a rectangular channel of dimensions $[0, 1.1] \times [0, 0.41]$ containing a circular cylinder of radius 0.05 centered at (0.2, 0.2)—rather than the unit-circle scaling used in Chapter 1 as a theoretical level-set example.

3.2 Navier-Stokes Equations

To simulate the background fluid in which our vesicle lives, we must solve the Navier-Stokes equations. These are the fundamental rules that describe how fluids flow.

We assume the fluid is "incompressible", which means its density does not change (like water or blood, unlike air). For such a fluid, the physics is governed by two main principles.

Continuity Equation

The first principle expresses conservation of mass. For an incompressible fluid, this reduces to:

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

where $\mathbf{u} = (u, v)$ is the velocity field.

Momentum Balance

The second principle corresponds to Newton's second law applied to a fluid element, expressed per unit mass. For an incompressible Newtonian fluid with constant viscosity and in the absence of body forces, the momentum equation reads:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \frac{1}{\rho}\nabla p - \nu \nabla^2 \mathbf{u} = \mathbf{0}. \tag{3.2}$$

Here, $\frac{\partial \mathbf{u}}{\partial t}$ is the unsteady acceleration, $(\mathbf{u} \cdot \nabla)\mathbf{u}$ is the convective term, $\frac{1}{\rho}\nabla p$ is the pressure force per unit mass, and $\nu \nabla^2 \mathbf{u}$ is the viscous diffusion term.

3.3 Mixed-Variable Formulation

To solve the Navier-Stokes equations, Rao et al. [17] propose a mixed-variable scheme. Rather than predicting velocity directly, the network outputs a stream function ψ, from which the velocity components are recovered analytically via Automatic Differentiation:

$$u = \frac{\partial \psi}{\partial y}, \qquad v = -\frac{\partial \psi}{\partial x}. \tag{3.3}$$

Because any velocity field derived this way satisfies $\nabla \cdot \mathbf{u} = 0$ identically, incompressibility is enforced exactly rather than approximately. In addition to ψ and the pressure p, the network also outputs the in-plane stress components $(\sigma_{11}, \sigma_{22}, \sigma_{12})$, which appear in the continuum form of the momentum equation:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v} = \frac{1}{\rho}\nabla \cdot \boldsymbol{\sigma} + \mathbf{g}, \qquad \boldsymbol{\sigma} = -p\mathbf{I} + \mu(\nabla \mathbf{v} + \nabla \mathbf{v}^T). \tag{3.4}$$

Casting the residual in terms of $\boldsymbol{\sigma}$ rather than directly in terms of second-order velocity derivatives reduces the highest derivative order in the loss function by one, which is the key trainability

advantage of the mixed-variable scheme. The full input–output map of the network is therefore:

$$[x,\, y,\, t] \xrightarrow{\mathcal{N}_\theta} [\psi,\, p,\, \sigma_{11},\, \sigma_{22},\, \sigma_{12}], \tag{3.5}$$

and the physical velocity (u, v) is recovered from ψ through equation (3.3).

3.4 Residuals

Once the network predicts the mixed-variable solution, we check how well it satisfies the governing equations by evaluating residual functions. Using Automatic Differentiation, the required derivatives are computed to machine precision.

Continuity

Because the stream-function formulation is used, mass conservation is satisfied automatically and contributes no residual term to the loss.

Momentum Residuals

These functions measure the violation of Newton's second law in the x- and y-directions:

$$e_2 = \frac{\partial u}{\partial t} + \left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) + \frac{\partial p}{\partial x} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{3.6}$$

$$e_3 = \frac{\partial v}{\partial t} + \left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) + \frac{\partial p}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{3.7}$$

The residuals e_2 and e_3 share the same parameter set θ as the primary network. If a weight is adjusted to correct the velocity field, the residuals are updated automatically through the chain rule, allowing physics to be enforced continuously during training.

3.5 Boundary Conditions as Data

In a PINN, boundary conditions are enforced as soft constraints: the network is penalized whenever it violates a boundary requirement, rather than having the value hard-coded as in a traditional solver.

For the no-slip condition on solid walls (channel walls and cylinder surface), a set of N_b boundary collocation points is sampled and the following loss term is minimized:

$$\mathcal{L}_{wall} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left(\|u(x_i, y_i)\|^2 + \|v(x_i, y_i)\|^2\right) \tag{3.8}$$

For the inlet, where the parabolic velocity profile is prescribed, an analogous data loss enforces $u(0, y) = u_{inlet}(y)$. The outlet condition imposes zero pressure, contributing to $\mathcal{L}_{outlet}$.

3.6 The Composite Loss Function

All contributions are combined into a single scalar total loss:

$$\mathcal{L}_{total} = J_g + \beta\, J_{i/bc} \tag{3.9}$$

where J_g is the governing-equation (PDE residual) loss evaluated at interior collocation points, $J_{i/bc}$ is the initial and boundary condition loss, and $\beta > 0$ is a user-defined weighting coefficient. Rao et al. perform a systematic study of β (their Figure 5) and find that the mixed-variable scheme yields consistent convergence across a wide range of values, whereas the traditional PINN scheme is highly sensitive to this choice. This robustness is one of the main advantages of the mixed-variable formulation.

3.7 Numerical Validation

Before extending the PINN framework to the coupled biomembrane problem introduced in Chapter 1, it is necessary to validate the implementation on the benchmark of Rao et al. This section and those that follow reproduce Figures 3–8 of the paper side-by-side with the present results so that agreements and discrepancies can be identified directly.

Key features of the benchmark

Two properties of the Rao et al. benchmark are worth emphasising. First, the network is trained with *no measurement data* of any kind. The only inputs to the training are the governing equations and the initial and boundary conditions. Pressure in particular is never prescribed; it is recovered entirely from the momentum PDE residual. Second, the mixed-variable formulation is shown by the authors to be markedly superior to the traditional $\{u, v, p\}$ PINN: the traditional scheme fails to enforce the no-slip condition on the channel walls, whereas the mixed-variable scheme reproduces the ANSYS Fluent reference with high accuracy.

Geometry and boundary conditions

The computational domain is the rectangular channel $[0,\, 1.1] \times [0,\, 0.41]\,\mathrm{m}$ containing a circular cylinder of radius 0.05 m centered at $(0.2,\, 0.2)\,\mathrm{m}$ (Figure 2 of Rao et al.). A parabolic velocity profile is imposed at the inlet, zero pressure at the outlet, and no-slip on all solid surfaces.

Network architecture and training

For the steady-state case, a grid search (Table 1 of Rao et al.) identifies the 8×40 network (8 hidden layers, 40 neurons each) as the best configuration, achieving a relative ℓ_2 velocity error of 1.8×10^{-2}. Training uses Adam followed by L-BFGS. For the transient case, a 7×50 network is used; the collocation set grows to $N_g = 120\,000$ interior, $N_{db} = 9\,600$ Dirichlet, $N_{nb} = 3\,200$ Neumann, and $N_I = 3\,500$ initial-condition points.

Present implementation

The steady case was reproduced by loading the converged checkpoint and evaluating it on the

Rao benchmark geometry. The transient case was retrained with an optimized TensorFlow 2 implementation using 2000 Adam iterations followed by 5000 L-BFGS-B iterations. Relative to the original TensorFlow 1 script, the optimized code also uses a modified loss weighting with a uniform boundary-condition weight $\beta = 2$. The transient comparison should therefore be read as a benchmark reproduction using an optimized variant of the original code rather than an exact duplication of every training choice. For scientific transparency, the benchmark implementation was based on the source code provided by Rao et al. for steady state case and also used it for unsteady however optimized the baseline code for the purposes of this study.

Table 3.1: Transient PINN loss components at the start of Adam training and after the full optimized schedule (2000 Adam + 5000 L-BFGS-B).

Loss component	Adam iteration 0	Final optimized run
Total $\mathcal{L}$	8.168×10^{-1}	6.050×10^{-3}
Governing equation $\mathcal{L}_f$	2.965×10^{-1}	3.983×10^{-3}
Initial condition $\mathcal{L}_{ic}$	6.906×10^{-2}	4.094×10^{-5}
Wall (no-slip) $\mathcal{L}_{wall}$	6.076×10^{-2}	8.484×10^{-4}
Inlet $\mathcal{L}_{inlet}$	1.178×10^{-1}	1.374×10^{-4}
Outlet $\mathcal{L}_{outlet}$	1.251×10^{-2}	6.826×10^{-6}

The total loss decreases from 8.168×10^{-1} to 6.050×10^{-3}, i.e. by about 99%. This is a substantial improvement over the earlier short transient draft, but the training horizon remains much shorter than the full schedule reported by Rao et al. The transient figures in Section 3.12 should therefore be interpreted as partially converged results: the global flow topology is reproduced well, whereas the transient pressure magnitude and timing still exhibit visible mismatch with the paper.

3.8 Reproduction of the Steady Benchmark

The collocation point distribution used for training the steady network is shown in Figure 3.1. Points are generated using Latin Hypercube Sampling and are refined near the cylinder surface to better resolve the boundary layer.

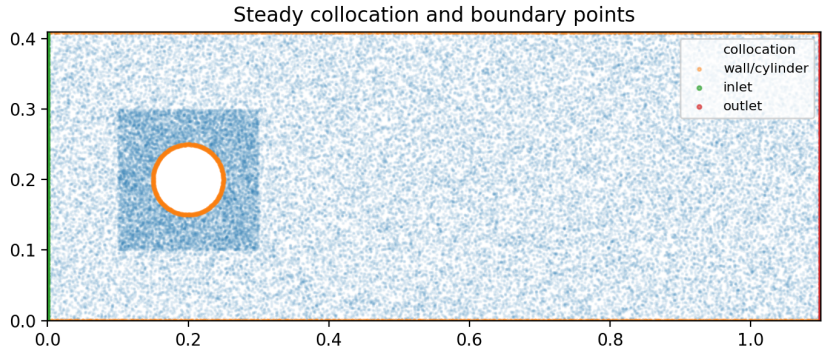
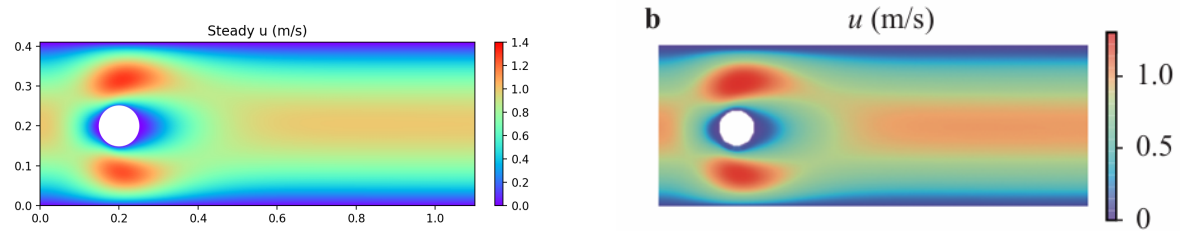


Figure 3.1: Collocation point distribution for the steady benchmark. Points are denser near the cylinder and walls to improve resolution of strong local gradients.

3.9 Steady-State Results

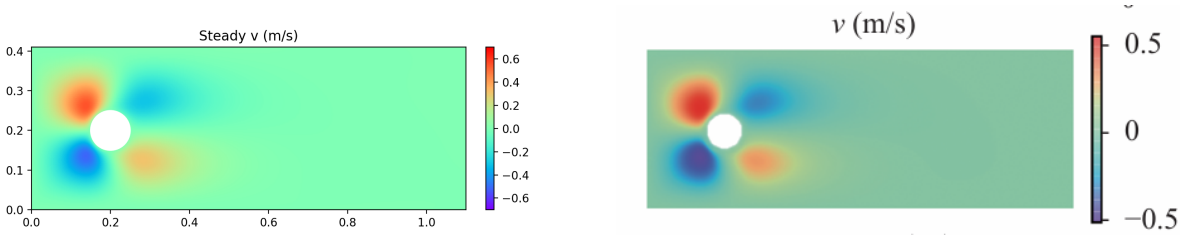
The full-set loss of the reimplemented network on the Rao benchmark geometry is $\mathcal{L} = 1.806 \times 10^{-4}$, with dominant contribution from the governing-equation residual ($\mathcal{L}_f = 1.715 \times 10^{-4}$). Boundary contributions are: wall 3.51×10^{-6}, inlet 8.02×10^{-7}, outlet 2.60×10^{-7}. These small values confirm that the loaded network accurately satisfies both the physical equations and the boundary conditions.

Figures 3.2–3.4 compare, for each field variable, the present result (left) against the Rao et al. mixed-variable result from Figure 3(b) of the paper (right). The results are in strong qualitative agreement across all three fields.



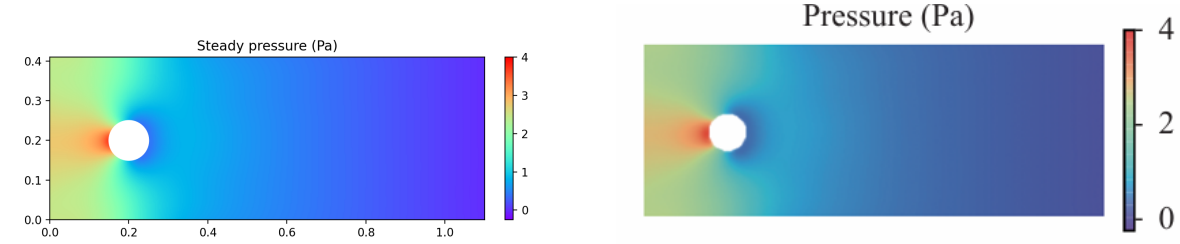
(a) Present result (b) Rao et al. Fig. 3(b)

Figure 3.2: Comparison of the streamwise velocity u (m/s) for the steady cylinder benchmark. The parabolic inlet profile, the velocity deficit in the wake, and the acceleration regions flanking the cylinder are all well reproduced.



(a) Present result (b) Rao et al. Fig. 3(b)

Figure 3.3: Comparison of the transverse velocity v (m/s) for the steady cylinder benchmark. The antisymmetric deflection pattern around the cylinder is correctly recovered.



(a) Present result (b) Rao et al. Fig. 3(b)

Figure 3.4: Comparison of the pressure field p (Pa) for the steady cylinder benchmark. Pressure is recovered entirely from the momentum PDE residual without any pressure training data. The stagnation pressure maximum at the upstream cylinder face and the low-pressure wake are both well captured.

For a more complete view, Figure 3.5 shows all three fields together in a single triplet panel, mirroring the layout of Figure 3 of the paper.

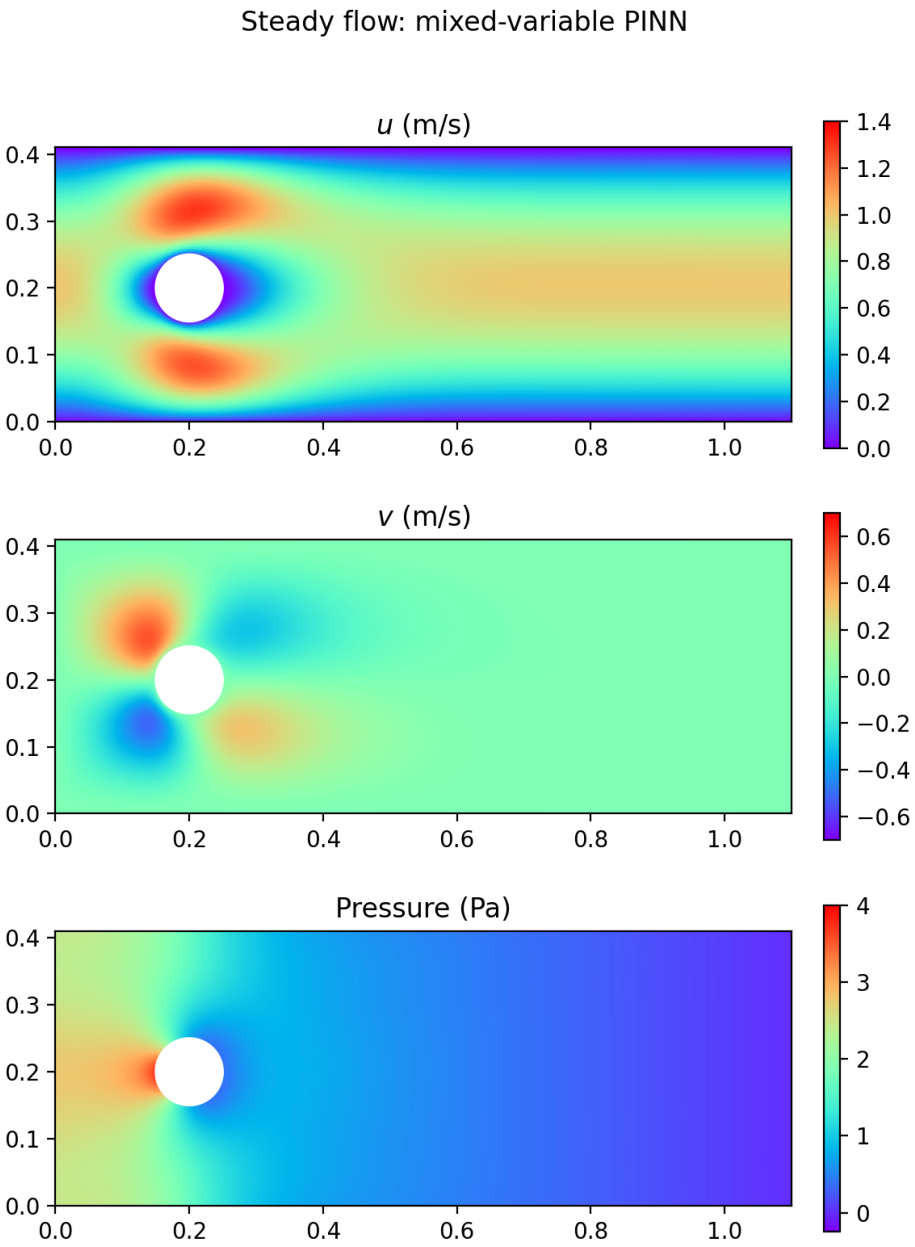


Figure 3.5: Combined triplet panel of the steady mixed-variable PINN prediction: streamwise velocity u (top), transverse velocity v (middle), and pressure p (bottom). Compare with Figure 3(b) of Rao et al.

The streamline pattern in Figure 3.6 provides an additional verification of the flow topology: smooth attached streamlines upstream and a symmetric wake directly behind the cylinder at this low Reynolds number.

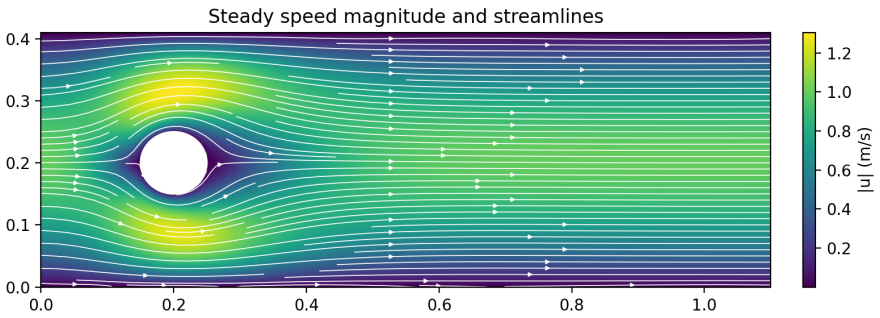
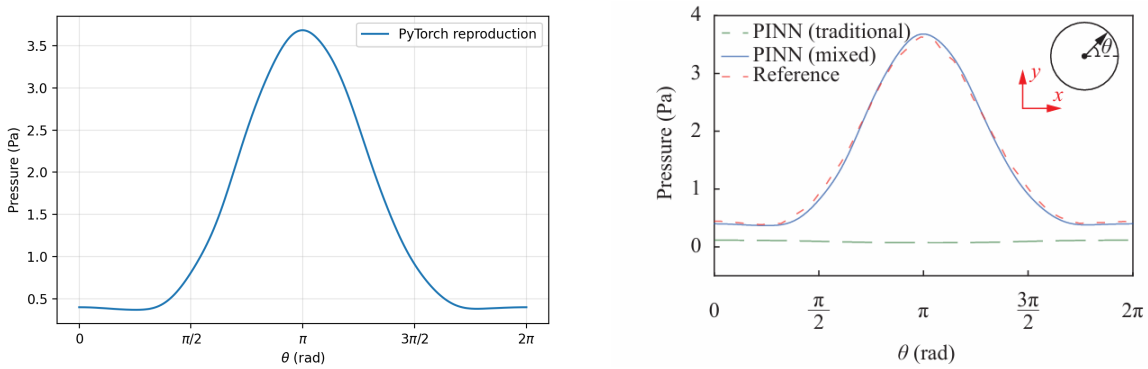


Figure 3.6: Streamlines of the steady PINN solution. Flow accelerates around the cylinder and forms a symmetric wake consistent with laminar flow at low Reynolds number.

Figure 3.7 compares the pressure distribution along the cylinder surface (polar angle θ) against Figure 4 of Rao et al. This is arguably the most important quantitative diagnostic because the pressure is recovered from the PDE residual and never imposed as a training target.



(a) Present result

(b) Rao et al. Fig. 4

Figure 3.7: Comparison of the pressure distribution on the cylinder surface as a function of polar angle θ. The stagnation-point maximum at $\theta \approx 0$ and the pressure minimum near the lateral positions are well reproduced, demonstrating accurate implicit pressure recovery from the PDE residual.

The steady-state training was performed locally on an Intel Core i7-8750H CPU. The total training time was approximately 10 hours. Since this runtime was noted from the local execution history rather than a saved training log, it should be regarded as an approximate computational cost. For the benchmark, the computational domain was $([0, 1.1] \times [0, 0.41])$, with fluid properties $(\rho = 1.0)$ and $(\mu = 0.02)$. The mixed-variable PINN used an 8 $(\times)$ 40 architecture, i.e. input dimension 2, eight hidden layers with 40 neurons each, and output dimension 5:

$$[[2] + 8 \times [40] + [5].]$$

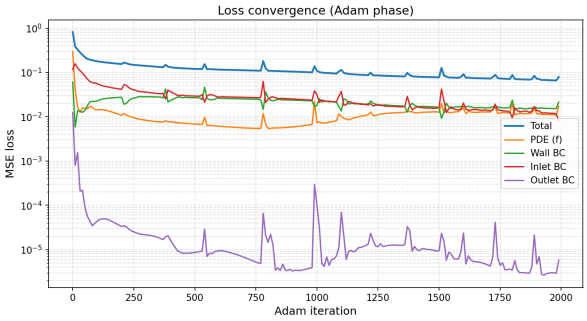
The inlet used a parabolic profile with $(U_{\max} = 1.0)$, the outlet imposed zero pressure, and no-slip conditions were enforced on the cylinder and channel walls. The loss was defined as

$$[\mathcal{L} = \mathcal{L} * f + 2(\mathcal{L} * wall + \mathcal{L} * inlet + \mathcal{L} * outlet).]$$

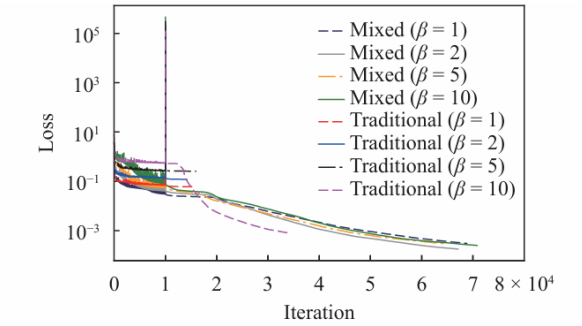
Training used 10000 Adam iterations with learning rate (5×10^{-4}), followed by L-BFGS-B optimization. The steady collocation set was generated by Latin hypercube sampling, with $40,000$ base interior points and $10,000$ refined points near the cylinder, together with boundary points on the wall, inlet, and outlet.

3.10 Convergence Comparison

Figure 3.8 compares the Adam-phase convergence curve from the present optimized transient run with the β-sensitivity study reported in Figure 5 of Rao et al. Because the left panel shows only the first 2000 Adam iterations, whereas the paper reports much longer Adam + L-BFGS schedules, the comparison is qualitative rather than iteration-for-iteration.



(a) Present optimized run (Adam phase, 2000 iterations)



(b) Rao et al. Fig. 5 (8×10^4 iterations)

Figure 3.8: Comparison of convergence behaviour for the transient benchmark. The present run shows stable loss reduction during the Adam stage, while the reference paper continues training much longer and reaches a lower loss level after a full Adam + L-BFGS schedule. Figure 5 of Rao et al. shows that the mixed-variable formulation has more stable convergence fpr different values in β, wehereas the traditional formualtion is more sensitive to this parameter. the sharp spike observed in some curves indicates temporary temporary optimzation instability, where the loss increases abruptly before the training stabilizes again. This supports the clam that the mixed-variable scheme is more trainable and robust.

Figure 3.8 shows that the optimized run already produces a clear downward trend during the Adam stage. After this stage, an additional 5000 L-BFGS-B iterations further reduce the final total loss to 6.050×10^{-3}. Rao et al., however, continue optimization to about 8×10^4 iterations and drive the loss into the 10^{-3} range. The present transient run is is relatively good but remains shorter and only partially converged relative to the reference study.

3.11 Reproduction of the Transient Benchmark

The transient benchmark uses the same channel-and-cylinder geometry but extends the PINN input to the full spatio-temporal domain (x, y, t) over the interval $t \in [0, 0.5]\,\mathrm{s}$. The dynamic viscosity is reduced to $\mu = 5 \times 10^{-3}\,\mathrm{kg\,(m\,s)^{-1}}$ while the density remains $\rho = 1\,\mathrm{kg\,m^{-3}}$. The inlet

boundary condition becomes time-dependent:

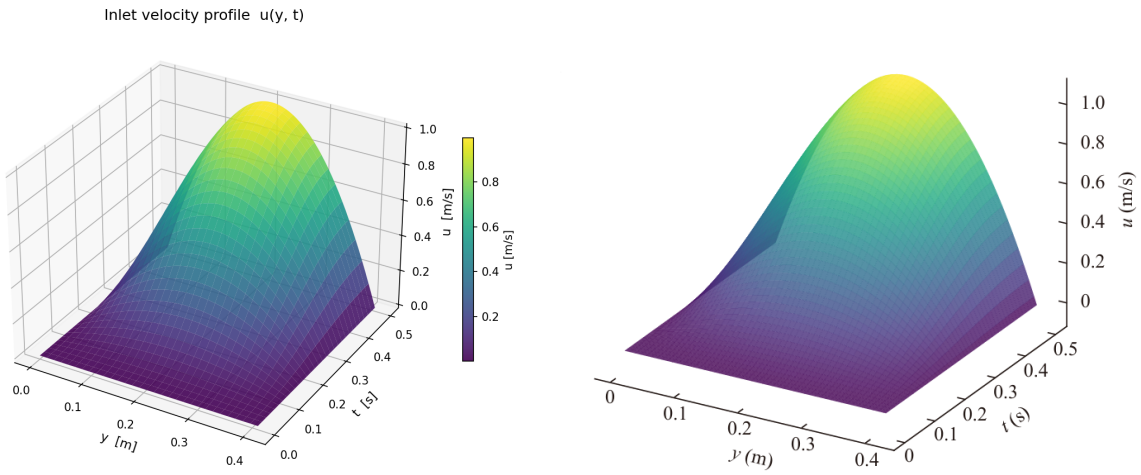
$$u(0,\, y,\, t) = 4\left[\sin\left(\frac{2\pi t}{T} + \frac{3\pi}{2}\right) + 1\right] U_{\max} \frac{(H-y)\, y}{H^2}, \tag{3.10}$$

with $U_{\max} = 0.5\,\mathrm{m/s}$ and period $T = 1.0\,\mathrm{s}$. Three pressure probes are monitored at $P_1 = (0.15,\, 0.2)\,\mathrm{m}$, $P_2 = (0.2,\, 0.25)\,\mathrm{m}$, and $P_3 = (0.25,\, 0.2)\,\mathrm{m}$ on the cylinder surface. The transient network has architecture 7×50.

3.12 Transient Results

Inlet profile (Figure 6 of Rao et al.)

Figure 3.9 compares the time-dependent inlet velocity profile prescribed by equation (3.10) with Figure 6 of the paper. The inlet amplitude varies sinusoidally with period $T = 1.0\,\mathrm{s}$, starting at zero, reaching its maximum near $t = 0.25\,\mathrm{s}$, and returning to zero at $t = 0.5\,\mathrm{s}$.



(a) Present result

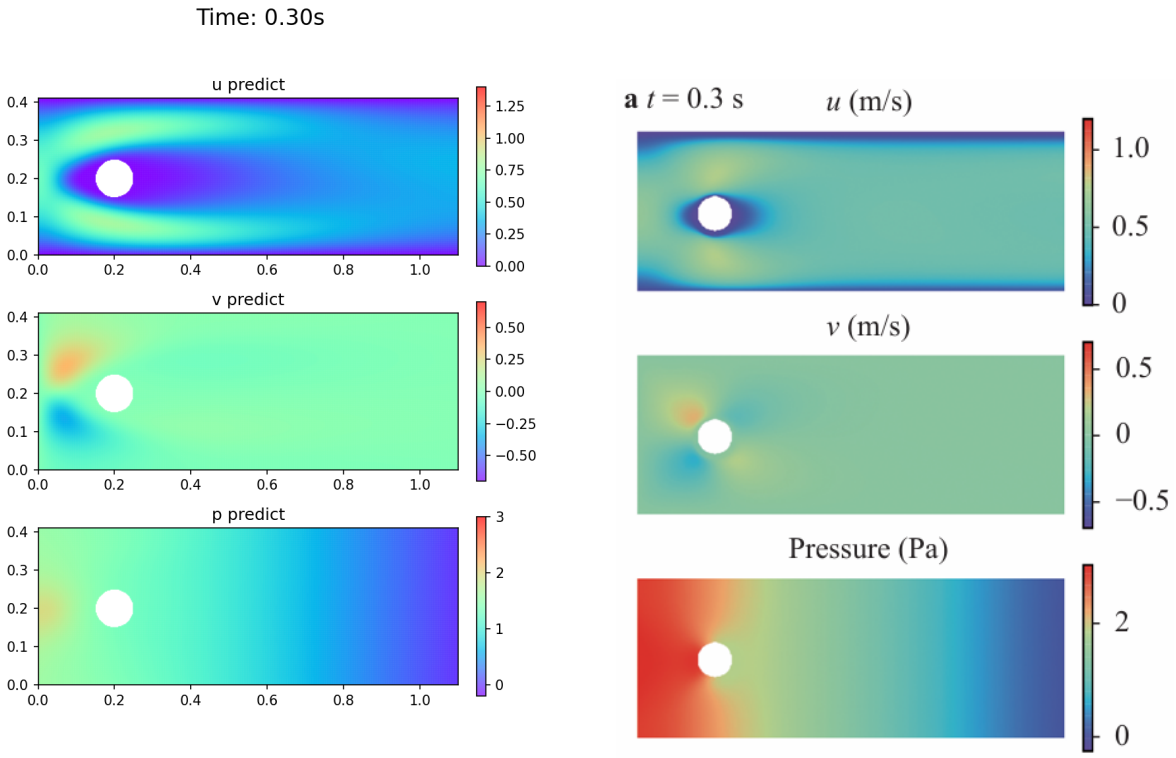
(b) Rao et al. Fig. 6

Figure 3.9: Comparison of the time-dependent inlet velocity profile $u(0, y, t)$. The present surface matches the prescribed boundary condition and agrees with the reference figure, confirming correct implementation of the transient inlet profile.

Figure 3.9 shows excellent agreement with the paper, as expected for a prescribed boundary condition. This confirms that the discrepancies seen later in Figures 3.10–3.13 are not caused by an incorrect inlet implementation.

Flow-field snapshots (Figure 7 of Rao et al.)

Figure 7 of the paper shows three snapshots at $t = 0.3$, 0.4, and 0.5 s of the streamwise velocity u, transverse velocity v, and pressure p. Figures 3.10–3.12 compare those paper snapshots with the corresponding results from the present optimized run.

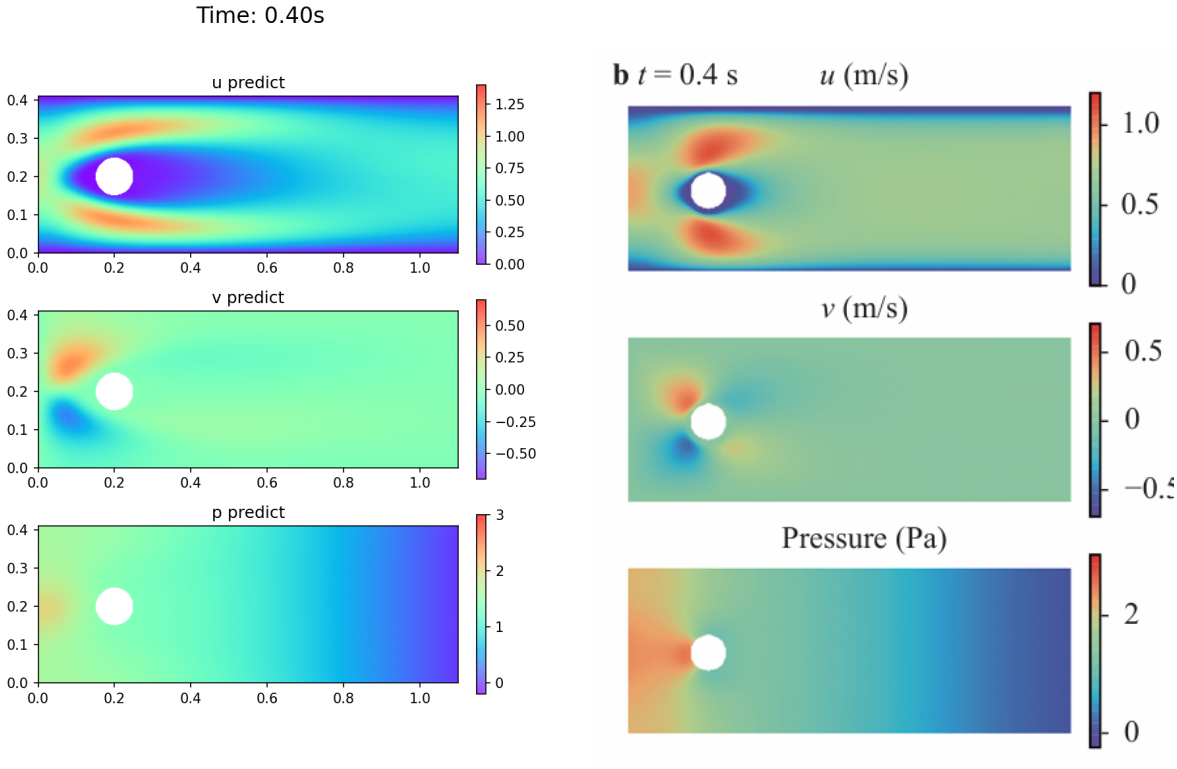


(a) Present optimized run ($t = 0.3\,\mathrm{s}$)

(b) Rao et al. Fig. 7(a), $t = 0.3\,\mathrm{s}$

Figure 3.10: Comparison of transient flow fields at $t = 0.3\,\mathrm{s}$. The present result reproduces the correct wake structure and the sign pattern of v, showing qualitative agreement with Rao et al., but the pressure field is smoother and the upstream pressure feature is less localized.

Figure 3.10 shows good qualitative agreement at $t = 0.3\,\mathrm{s}$: the accelerated flow around the cylinder, the wake deficit, and the antisymmetric v pattern are all captured. The main discrepancy lies in the pressure field, which is more diffuse than in Rao et al. and does not reproduce the sharp local pressure feature at the upstream face with the same intensity.

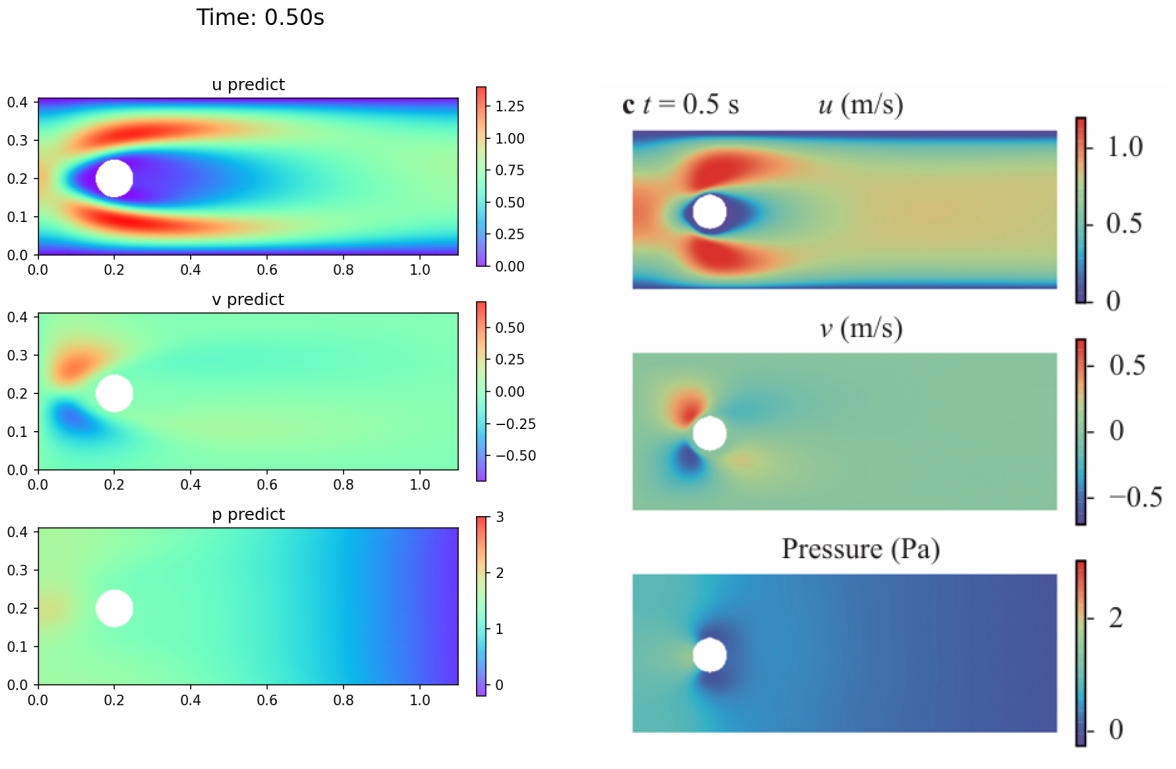


(a) Present optimized run ($t = 0.4\,\mathrm{s}$)

(b) Rao et al. Fig. 7(b), $t = 0.4\,\mathrm{s}$

Figure 3.11: Comparison of transient flow fields at $t = 0.4\,\mathrm{s}$. The velocity contours remain in qualitative agreement with the paper, especially the upper and lower acceleration lobes around the cylinder, whereas the predicted pressure gradient is still more diffuse than in the reference plot.

Figure 3.11 preserves the same overall topology at $t = 0.4\,\mathrm{s}$, and the upper and lower high-speed lobes in u are positioned correctly. Nevertheless, the recovered pressure remains too diffuse, so the agreement is better for the velocity field than for the pressure field.



(a) Present optimized run ($t = 0.5\,\mathrm{s}$)

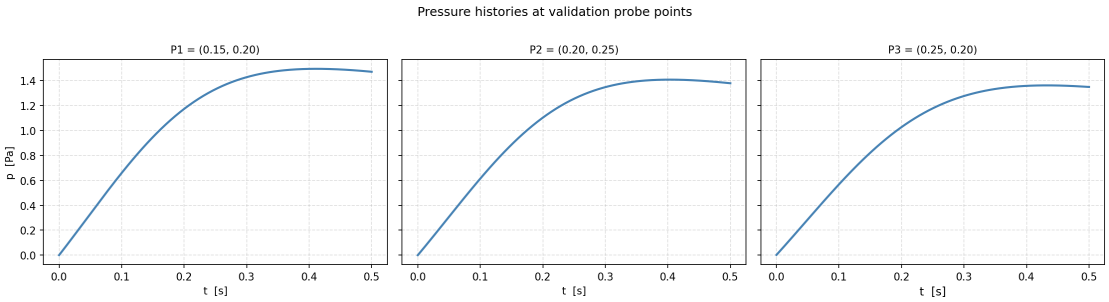
(b) Rao et al. Fig. 7(c), $t = 0.5\,\mathrm{s}$

Figure 3.12: Comparison of transient flow fields at $t = 0.5\,\mathrm{s}$. A clearer qualitative mismatch appears: the present run overpredicts the streamwise velocity near the cylinder and does not recover the sharp localized pressure structure reported by Rao et al.

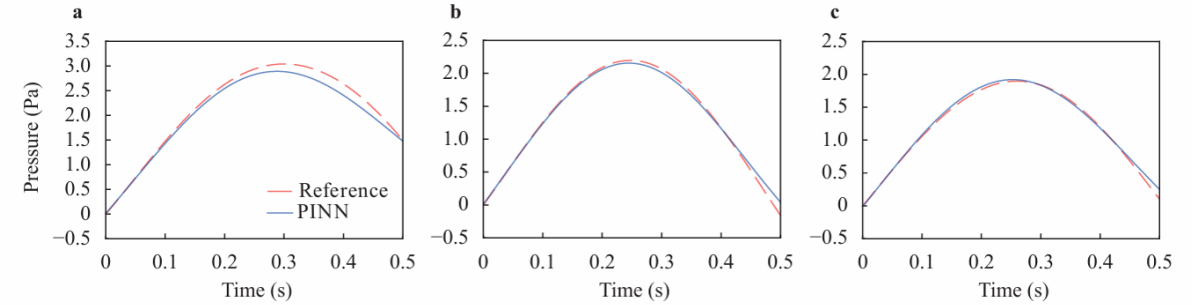
Figure 3.12 reveals the most visible qualitative mismatch. The present solution overpredicts the streamwise velocity near the cylinder and misses the sharp localized pressure structure reported in the paper. This behaviour is consistent with partial convergence of the transient PINN and with the modified loss weighting of the optimized code, which changes how strongly the wall and inlet constraints are enforced.

Pressure probe histories (Figure 8 of Rao et al.)

Figure 3.13 compares the pressure time histories at the three cylinder-surface probes P_1, P_2, and P_3 with Figure 8 of the paper.



(a) Present result



(b) Rao et al. Fig. 8

Figure 3.13: Comparison of pressure histories at the cylinder-surface probes P_1, P_2, and P_3. The present result reproduces the correct probe ordering and smooth temporal evolution, but it underestimates the peak pressure and delays the decay phase relative to Rao et al., indicating qualitative mismatch.

Figure 3.13 still captures two correct qualitative features: all curves start from zero, and the probe ordering $P_1 > P_2 > P_3$ is preserved throughout the cycle. However, the mismatch with the paper is clear. The peak pressures are substantially lower than in Rao et al., the maxima occur too late, and the decay toward $t = 0.5\,\mathrm{s}$ is too weak. Since pressure is recovered only indirectly through the momentum residual, these errors indicate that the transient pressure field is less converged than the velocity field; the shorter Adam/L-BFGS-B iterations and modified loss weighting are the most plausible causes for the mismatch. For the optimized transient benchmark, the computational domain was

$$[0, 1.1] \times [0, 0.41] \times [0, 0.5],$$

with fluid properties

$$\rho = 1.0, \qquad \mu = 0.005.$$

The network used a 7×50 architecture, i.e.

$$[3] + 7 \times [50] + [5].$$

The optimized run used 2000 Adam iterations followed by 5000 L-BFGS-B iterations. The final loss was

$$6.050 \times 10^{-3}.$$

The total loss was defined as

$$\mathcal{L}_{\text{total}} = \mathcal{L}_f + \beta\left(\mathcal{L}_{\text{WALL}} + \mathcal{L}_{\text{INLET}} + \mathcal{L}_{\text{OUTLET}} + \mathcal{L}_{\text{IC}}\right), \qquad \beta = 2.$$

The collocation set contained 140,799 points after preprocessing and removing points inside the cylinder. The raw sampling consisted of 90,000 base interior points, 30,000 refined points near the cylinder, and 3,000 points in each near-wall refinement band, together with 10,773 wall points, 3,721 inlet points, 3,321 outlet points, and 3,265 initial-condition points. On an NVIDIA T4 GPU (15 GB VRAM), the optimized transient run converged to a final loss of 6.050×10^3 in 2609.6 s (≈ 43.5 min).

3.13 Comparison with Traditional Methods

The benchmark study of Rao et al. provides a direct comparison between the mixed-variable PINN, the traditional PINN, and the finite-volume reference solution from ANSYS Fluent.

Mesh dependency. The Finite Volume Method (FVM) requires a structured or unstructured mesh, refined near the cylinder to resolve the boundary layer. Coarse meshes introduce artificial dissipation that smears the wake. The PINN replaces the mesh with collocation points that can be distributed freely and refined near the obstacle simply by adjusting the sampling density, without re-meshing.

Incompressibility enforcement. In both FVM and the traditional PINN, incompressibility is enforced approximately—via a pressure-correction step in FVM, or as an additional residual term in the loss in the traditional PINN. In the mixed-variable PINN, the stream-function output guarantees exact incompressibility by construction, removing the continuity residual from the loss entirely.

Pressure recovery. In FVM, pressure is computed simultaneously with velocity by the coupled solver. In the mixed-variable PINN, pressure is never prescribed at any boundary except the outlet; it is recovered entirely from the momentum PDE residual. The close agreement of the cylinder-surface pressure curve in Figure 3.7 with the ANSYS Fluent reference demonstrates that this implicit recovery is accurate.

Derivative order in the loss. The traditional PINN loss requires second-order velocity derivatives from the network output. The mixed-variable formulation, by introducing the stress components $\boldsymbol{\sigma}$ as additional outputs, reduces the derivative order of the residuals and substantially improves trainability, as demonstrated by the convergence curves in Figure 3.8.

3.14 Continuous Time Modeling

One of the most significant differences between the PINN approach and classical CFD lies in the treatment of time.

Classical time-stepping. In a traditional Finite Element or Finite Volume solver, time is

treated sequentially. To find the solution at t_{n+1}, the solver must know the solution at t_n. This requires an iterative time-stepping loop (e.g., Runge-Kutta or Crank-Nicolson), errors from previous steps can accumulate, and there is a CFL stability condition that limits the maximum allowable time step.

The spatio-temporal domain. In the PINN framework, time t is simply an additional input coordinate on equal footing with x and y. The neural network views the entire spatio-temporal domain $\Omega \times [0, T]$ as a single hypercube:

$$\text{Input:} \quad \{x,\, y,\, t\} \quad \longrightarrow \quad \text{Output:} \quad \{\psi,\, p,\, \sigma_{11},\, \sigma_{22},\, \sigma_{12}\} \tag{3.11}$$

This leads to three unique characteristics: (1) the network learns the flow at $t = 0$ and $t = 0.5\,\text{s}$ simultaneously during training; (2) the solution can be queried at any arbitrary instant without interpolating between discrete time steps; and (3) there is no CFL condition, because the network does not advance the solution in time.

Conclusion

This project investigated the use of Physics-Informed Neural Networks (PINNs) for the approximation of incompressible fluid flows. The PINN framework was introduced and applied to the Navier–Stokes equations using the mixed-variable formulation, and evaluated on standard benchmark problems involving flow past a circular cylinder.

The numerical results show that the method is able to reproduce some of the main qualitative features of the flow, such as the wake structure and general velocity patterns, in both steady and transient regimes. However, the agreement with reference solutions remains primarily qualitative. In particular, noticeable discrepancies were observed in the pressure field, which appears more diffuse and underestimates peak values, as well as in the transient evolution reflecting incomplete convergence and the sensitivity of the method to the training procedure.

These limitations highlight that the present implementation should be regarded as a preliminary validation rather than a fully quantitative study. The accuracy of the results is affected by several factors, including the relatively coarse spatial and temporal resolution, the limited training duration, and the high computational cost, with training times reaching several hours even for steady configurations.

Future work should focus on improving both accuracy and efficiency. This includes increasing resolution, optimizing the training strategy and computational cost, refining the loss weighting, and exploring adaptive sampling techniques. A more systematic validation using quantitative error metrics and additional benchmark cases is also necessary. Finally, extending the approach to the original multiphysics objective of the project, namely the coupling of incompressible flow with membrane dynamics presented in the first chapter, remains an important but challenging direction for further research.

Bibliography

[1] J. W. Barrett, H. Garcke, and R. Nürnberg, "Numerical computations of the dynamics of fluidic membranes and vesicles", *Physical Review E*, vol. 92, no. 5, p. 052704, 2015.

[2] J. Beaucourt, F. Rioual, T. Séon, T. Biben, and C. Misbah, "Steady to unsteady dynamics of a vesicle in a flow", *Physical Review E*, vol. 69, p. 011906, 2004.

[3] G. Dziuk, "Computational parametric Willmore flow", *Numerische Mathematik*, vol. 111, no. 1, pp. 55–80, 2008.

[4] W. Helfrich, "Elastic properties of lipid bilayers: theory and possible experiments", *Zeitschrift für Naturforschung C*, vol. 28, pp. 693–703, 1973.

[5] E. M. Kolahdouz and D. Salac, "A numerical model for the trans-membrane voltage of vesicles", *Applied Mathematics Letters*, vol. 39, pp. 7–12, 2015.

[6] A. Laadhari, "An operator splitting strategy for fluid-structure interaction problems with thin elastic structures in an incompressible Newtonian flow", *Applied Mathematics Letters*, vol. 81, pp. 35–43, 2018.

[7] A. Laadhari, "Exact Newton method with third-order convergence to model the dynamics of bubbles in incompressible flow", *Applied Mathematics Letters*, vol. 69, pp. 138–145, 2017.

[8] A. Laadhari, "Implicit finite element methodology for the numerical modeling of incompressible two-fluid flows with moving elastic interface", *Applied Mathematics and Computation*, vol. 333, pp. 376–400, 2018.

[9] A. Laadhari, "Stabilized weak-gradient discontinuous finite elements with optimal error estimates for second-order elliptic PDEs", *Mathematical Modelling and Numerical Simulation with Applications*, vol. 5, no. 3, pp. 644–680, 2025.

[10] A. Laadhari, P. Saramito, and C. Misbah, "An adaptive finite element method for the modeling of the equilibrium of red blood cells", *International Journal for Numerical Methods in Fluids*, vol. 80, no. 7, pp. 397–428, 2016.

[11] A. Laadhari, P. Saramito, and C. Misbah, "Computing the dynamics of biomembranes by combining conservative level set and adaptive finite element methods", *Journal of Computational Physics*, vol. 263, pp. 328–352, 2014.

[12] A. Laadhari, P. Saramito, and C. Misbah, "Improving the mass conservation of the level set method in a finite element context", *Comptes Rendus Mathématique*, vol. 348, no. 9, pp. 535–540, 2010.

[13] A. Laadhari, P. Saramito, C. Misbah, and G. Székely, "Fully implicit method for the dynamics of biomembranes and capillary interfaces by combining the level set and Newton methods", *Journal of Computational Physics*, vol. 343, pp. 271–299, 2017.

[14] A. Laadhari, C. Misbah, and P. Saramito, "On the equilibrium equation for a generalized biological membrane energy by using a shape optimization approach", *Physica D: Nonlinear Phenomena*, vol. 239, pp. 1567–1572, 2010.

[15] A. Laadhari and G. Székely, "Fully implicit finite element method for the modeling of free surface flows with surface tension effect", *International Journal for Numerical Methods in Engineering*, vol. 111, no. 11, pp. 1074–1097, 2017.

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

[17] C. Rao, H. Sun, and Y. Liu, "Physics-informed deep learning for incompressible laminar flows", *Theoretical and Applied Mechanics Letters*, vol. 10, pp. 207–212, 2020.

[18] D. Salac and M. Miksis, "A level set projection model of lipid vesicles in general flows", *Journal of Computational Physics*, vol. 230, no. 22, pp. 8192–8215, 2011.

[19] J. A. Sethian and P. Smereka, "Level set methods for fluid interfaces", *Annual Review of Fluid Mechanics*, vol. 35, no. 1, pp. 341–372, 2003.