Solving Navier-Stokes Equation Using Physics-Informed Neural Networks (PINNs)

Project Overview:

- Duration - 3 months
- Project Mentor - Ninad Shetty
- Team Members - Ashish Manash
  Jasim
  Rakshitha Gowda
  Shreya Ravi

1. Introduction to the Navier-Stokes Equations

1.1 What Are the Navier-Stokes Equations?

The **Navier-Stokes equations** (NSE) describe the motion of fluid substances, such as air and water. They are based on:

- **Newton's Second Law** (Force = Mass × Acceleration) applied to fluid elements.
- **Conservation of Mass** (Continuity Equation).
- **Conservation of Momentum** (Momentum Equation).

The equations apply to both **laminar and turbulent** flows, but solving them becomes challenging at high Reynolds numbers.

1.2 Why Are They Important?

- **Predicting weather patterns** (atmospheric fluid dynamics).
- **Simulating blood flow** in arteries for medical research.

- **Designing aircraft and cars** (aerodynamics).
- **Studying ocean currents** and planetary atmospheres.
- **Optimizing industrial processes** (combustion in engines).

---

2. Mathematical Formulation of the Navier-Stokes Equations

2.1 General Form of the Navier-Stokes Equations

For an **incompressible, Newtonian fluid**, the Navier-Stokes equations are:

1. Continuity Equation (Mass Conservation)

$$\nabla \cdot \mathbf{u} = 0$$

where u=(u,v,w) is the velocity field in three dimensions. This equation ensures that mass is conserved, meaning fluid cannot spontaneously appear or disappear.

2. Momentum Equation (Newton's Second Law Applied to Fluids)

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

where:

- u=(u,v,w) → Velocity field.
- $\frac{\partial \mathbf{u}}{\partial t}$ → Time-dependent change in velocity.
- $(\mathbf{u} \cdot \nabla)\mathbf{u}$ → Convective acceleration (advection term).
- $-\frac{1}{\rho}\nabla p$ → Pressure gradient force.

- $\nu\nabla^2\mathbf{u}$ → Viscous term (diffusion).
- f → External forces (e.g., gravity, magnetic forces).

2.2 Navier-Stokes Equations in Two Dimensions (2D)

For a 2D incompressible fluid flow (x,yx, yx,y directions only), the equations reduce to:

Continuity Equation (2D Incompressible)

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

Momentum Equations (2D)

X-Momentum Equation

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

Y-Momentum Equation

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

---

3. Physical Interpretation of Terms

1. Unsteady Term $(\frac{\partial \mathbf{u}}{\partial t})$

Represents the **change in velocity** over time. If the flow is steady, this term is **zero**.

2. Convective Term $((\mathbf{u}\cdot\nabla)\mathbf{u})$

Describes how fluid **self-transports its own momentum**. This is a **nonlinear** term responsible for **turbulence**.

3. Pressure Gradient Term $(-\frac{1}{\rho}\nabla p)$

Represents how **fluid accelerates due to pressure differences**.

4. Viscous Diffusion Term $(\nu\nabla^2\mathbf{u})$

Accounts for **viscous forces**, which cause fluid to resist motion and **smooth out velocity differences**.

5. External Force Term (f)

Represents additional forces acting on the fluid (e.g., **gravity, electromagnetic forces**).

---

4. Challenges in Solving Navier-Stokes Equations

The **Navier-Stokes equations are difficult to solve** because:

- They are **nonlinear PDEs** (due to the convective term $(u \cdot \nabla)u$)
- Solutions can exhibit **turbulence**, making them unpredictable at high Reynolds numbers.
- They require **numerical methods** for real-world problems.

---

5. Numerical Methods for Solving Navier-Stokes Equations

5.1 Finite Difference Method (FDM)

- Uses **discrete points** on a grid to approximate derivatives.

- Requires **small time steps** for stability.

5.2 Finite Element Method (FEM)

- Divides the fluid domain into **small elements**.
- Solves equations locally and combines solutions.

5.3 Spectral Methods

- Uses **Fourier series or Chebyshev polynomials** to represent solutions.
- Provides **high accuracy** but requires **smooth solutions**.

5.4 Computational Fluid Dynamics (CFD) Software

Tools like **ANSYS Fluent, OpenFOAM, and COMSOL** solve the Navier-Stokes equations using **mesh-based methods**.

---

6. Special Cases of Navier-Stokes Equations

6.1 Euler Equations (When Viscosity = 0)

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p$$

- Used for **high-speed flows (e.g., supersonic jets)**.

6.2 Stokes Equations (When Inertia is Negligible)

$$0 = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u}$$

Used for **creeping flows (e.g., blood flow in capillaries)**.

1. Introduction to Physics-Informed Neural Networks (PINNs)

1.1 What Are PINNs?

Physics-Informed Neural Networks (**PINNs**) are a **deep learning-based framework** for solving **partial differential equations (PDEs)** by incorporating **physical laws** into the neural network's training process.

Unlike traditional neural networks, which rely solely on **data**, PINNs **embed physics constraints** (such as conservation laws) directly into the loss function. This makes PINNs a **data-efficient and interpretable approach** for solving complex scientific and engineering problems.

1.2 How Are PINNs Different from Traditional Solvers?

Most numerical methods for solving PDEs, such as the **Finite Difference Method (FDM), Finite Element Method (FEM), and Computational Fluid Dynamics (CFD) techniques**, rely on **discretizing the domain** into small elements or grids and solving the equations numerically. These approaches are:

- **Computationally expensive** for high-dimensional problems.
- **Require extensive meshing** (which can be difficult for complex geometries).
- **Sensitive to discretization errors**.

PINNs, in contrast:

- Use a **neural network to approximate the solution**.
- Do **not require mesh generation**.
- Can handle **high-dimensional problems efficiently**.

- Use **automatic differentiation** to compute gradients instead of numerical differentiation.

---

2. Why Use PINNs for Solving the Navier-Stokes Equations?

The **Navier-Stokes Equations (NSE)** govern fluid flow and are highly **nonlinear, coupled PDEs**. Traditional numerical solvers for NSE often face challenges, including:

- **High computational cost** (especially for turbulence and high Reynolds numbers).
- **Difficulty handling complex geometries**.
- **Instabilities in high-speed or multiphase flows**.

Using **PINNs** offers several advantages:

2.1 No Need for Discretization

- Traditional CFD methods require **meshing** (discretizing the space into a grid).
- PINNs do not rely on meshing; instead, they solve the problem in **continuous space**.

2.2 Handling High-Dimensional and Complex Geometries

- PINNs can be extended to **3D** or complex geometries without requiring complex meshing.
- They work well in problems where traditional methods struggle due to **mesh deformation**.

2.3 Efficiency in Solving Inverse Problems

- PINNs can **learn unknown parameters** (such as boundary conditions, source terms, or even missing physics) by solving **inverse problems**.
- For example, if we have **limited flow data**, PINNs can reconstruct the missing parts.

2.4 Automatic Differentiation for Computing PDE Residuals

- Traditional solvers use **finite difference approximations** to compute derivatives, which introduce truncation errors.
- PINNs use **automatic differentiation (AD)**, which provides highly accurate derivatives.

2.5 Solving Forward and Inverse Problems Simultaneously

- A **forward problem** solves for velocity and pressure given boundary conditions.
- An **inverse problem** finds unknown parameters (like viscosity or source terms).
- PINNs can handle **both types of problems in a unified framework**.

---

3. Formulation of PINNs for the Navier-Stokes Equations

To solve the **Navier-Stokes equations** using **PINNs**, we define a **neural network** that approximates the solution:

$$f_\theta(x, y, t) = [u(x, y, t), v(x, y, t), p(x, y, t)]$$

where:

- u,v → Velocity components in **x** and **y** directions.
- p → Pressure.
- θ → Neural network parameters (weights and biases).

The neural network learns to satisfy:

1. **The Governing Equations (Navier-Stokes PDEs)**
2. **The Incompressibility Condition**
3. **The Boundary Conditions**

3.1 Loss Function for PINNs

The loss function consists of **multiple components**:

(1) Governing Equation Residuals (Navier-Stokes Residual)

We enforce the **Navier-Stokes equations** as a soft constraint:

$$\mathcal{L}_{\text{NS}} = \left\| \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \right\|^2$$

$$\mathcal{L}_{\text{NS}} = \left\| \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \right\|^2$$

$$\mathcal{L}_{\text{continuity}} = \left\| \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right\|^2$$

(2) Boundary Condition Loss

Boundary conditions are enforced as:

$$\mathcal{L}_{\text{BC}} = \sum_{\text{boundary points}} \left(u_{\text{predicted}} - u_{\text{boundary}}\right)^2 + \left(v_{\text{predicted}} - v_{\text{boundary}}\right)^2$$

(3) Data Loss (If Experimental or Simulation Data Exists)

If we have real-world data points, we minimize:

$$\mathcal{L}_{\text{data}} = \sum_{i=1}^{N} \left(u_{\text{predicted}}(x_i, y_i) - u_{\text{measured}}(x_i, y_i)\right)^2$$

Total Loss Function

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{NS}} + \lambda_2 \mathcal{L}_{\text{continuity}} + \lambda_3 \mathcal{L}_{\text{BC}} + \lambda_4 \mathcal{L}_{\text{data}}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights that balance different terms.

---

4. Training the PINN Model to solve NSE:

Training the PINN The model is trained using the Adam optimizer, minimizing the loss function to find the best solution for the Navier-Stokes equations.
Algorithm

Step 1: Initialize the neural network with multiple layers and Tanh activation.
Step 2: Define input domain points and boundary conditions.
Step 3: Compute PDE residuals using automatic differentiation.
Step 4: Construct a loss function combining PDE residuals and boundary losses.
Step 5: Train the network using gradient-based optimization.
Step 6: Evaluate results and compare with numerical solutions.

---

5. Advantages and Limitations of PINNs

Advantages

✅ Can solve problems **without discretization**.
✅ Can handle **inverse problems**.
✅ Suitable for **high-dimensional PDEs**.
✅ Uses **automatic differentiation**, avoiding numerical errors.

Limitations

❌ Training PINNs can be **slow** for complex problems.
❌ **Hyperparameter tuning** (network architecture, optimizer) is crucial.
❌ **Gradient pathologies** may occur for high-Reynolds-number flows.

Physics-Informed Neural Networks (PINNs) provide a **promising alternative** to traditional solvers for the Navier-Stokes equations. They enable **data-driven and physics-constrained learning**, making them **ideal for fluid flow problems** like the **lid-driven cavity problem**.

Lid-Driven Cavity Problem: A Comprehensive Explanation

The **lid-driven cavity problem** is one of the most fundamental test cases in **computational fluid dynamics (CFD)**. It is widely used to study **incompressible, viscous fluid flow** and evaluate numerical methods for solving the **Navier-Stokes equations**.

This document provides a **detailed explanation** of the problem, including **physical description, governing equations, boundary conditions, flow behavior at different Reynolds numbers, numerical solutions, and real-world applications**.

---

1. Introduction to the Lid-Driven Cavity Problem

1.1 What Is the Lid-Driven Cavity Problem?

The **lid-driven cavity problem** consists of a **square or rectangular cavity filled with fluid**, where the **top boundary (lid) moves with a constant velocity** while the other three walls remain **stationary (no-slip condition)**. The moving lid **drives the fluid motion**, creating a primary vortex and **secondary vortices near the corners**.

This problem is used in:

- **Benchmarking numerical solvers** for Navier-Stokes equations.
- **Studying vortex formation and flow instabilities**.
- **Understanding the effects of different Reynolds numbers**.
- **Developing new numerical algorithms for CFD applications**.

1.2 Why Is It Important?

- **Simple geometry but complex physics**: Despite its simplicity, the problem exhibits **rich flow structures**, making it an excellent test case.
- **Turbulence and secondary vortices**: At higher Reynolds numbers, the flow becomes **unsteady and turbulent**, requiring advanced numerical methods.
- **Validating numerical solvers**: Since the problem has been extensively studied, it is **widely used to verify computational fluid dynamics (CFD) codes**.

---

2. Boundary Conditions

The problem is defined by **four boundaries**:

1. **Top Lid (Moving Wall):**
   $$u = U_{\text{lid}}, \quad v = 0$$
   The lid moves at a constant velocity, driving the flow.

2. **Left Wall (No-Slip Condition):**
   $$u = 0, \quad v = 0$$
   The velocity is **zero** at the wall (fluid sticks to the boundary).

3. **Right Wall (No-Slip Condition):**
   $$u = 0, \quad v = 0$$
   Similar to the left wall.

4. **Bottom Wall (No-Slip Condition):**
   $$u = 0, \quad v = 0$$
   The velocity is also **zero** at the bottom wall.

---

3. Flow Behavior and Reynolds Number Effects

The behavior of the **lid-driven cavity flow** is heavily influenced by the **Reynolds number (ReReRe)**, which is given by:

$$Re = \frac{U_{\text{lid}} L}{\nu}$$

where:

- L = cavity length.
- Ulid = lid velocity.
- v = kinematic viscosity.

3.1 Low Reynolds Number (Re < 100) - Laminar Flow

- The flow is **smooth and steady**.
- A **single, large primary vortex** is formed in the center.

- The fluid near the walls **moves very slowly due to viscosity**.

3.2 Intermediate Reynolds Number (100 < Re < 1000)

- **Secondary vortices** appear in the bottom corners.
- Flow **remains steady but becomes more complex**.

3.3 High Reynolds Number (Re > 3000) - Turbulence Onset

- The flow becomes **unsteady and turbulent**.
- **Multiple vortices form**, interacting chaotically.
- Strong **shear layers and recirculation zones** appear.

3.4 Very High Reynolds Number (Re > 10,000) - Fully Turbulent

- Flow **fluctuates randomly**.
- **Strong secondary and tertiary vortices** develop.
- **Pressure gradients become highly complex**.

---

5. Results of running code and Visualization

This code implements a **Physics-Informed Neural Network (PINN)** using PyTorch to solve fluid dynamics equations (Navier-Stokes equations). The model learns velocity (u,v) and pressure (p) fields within a domain while satisfying governing physical laws. The training process optimizes the neural network to minimize both equation residuals and boundary conditions.

1. **Defining PINN Model:**

```
class PINN(nn.Module):
    def __init__(self, layers):
        super(PINN, self).__init__()
        self.fc = nn.Sequential()
        for i in range(len(layers) - 1):
            self.fc.add_module(f'layer_{i}', nn.Linear(layers[i], layers[i + 1]))
            if i < len(layers) - 2:
                self.fc.add_module(f'activation_{i}', nn.Tanh())

    def forward(self, x):
        return self.fc(x)
```

A **fully connected neural network** (FCNN) is created.

The network structure is **defined dynamically** based on the `layers` parameter.

Tanh activation function is used in hidden layers to introduce non-linearity.

The output is a **3D vector** representing [u,v,p][u, v, p][u,v,p].

2. Computing PDE Residuals:

```
    def compute_residuals(model, x, y):
uvp = model(torch.cat([x, y], dim=1))
u, v, p = uvp[:, 0:1], uvp[:, 1:2], uvp[:, 2:3]

u_x = torch.autograd.grad(u, x, torch.ones_like(u), create_graph=True)[0]
u_y = torch.autograd.grad(u, y, torch.ones_like(u), create_graph=True)[0]
v_x = torch.autograd.grad(v, x, torch.ones_like(v), create_graph=True)[0]
v_y = torch.autograd.grad(v, y, torch.ones_like(v), create_graph=True)[0]
p_x = torch.autograd.grad(p, x, torch.ones_like(p), create_graph=True)[0]
p_y = torch.autograd.grad(p, y, torch.ones_like(p), create_graph=True)[0]

u_xx = torch.autograd.grad(u_x, x, torch.ones_like(u_x), create_graph=True)[0]
u_yy = torch.autograd.grad(u_y, y, torch.ones_like(u_y), create_graph=True)[0]
v_xx = torch.autograd.grad(v_x, x, torch.ones_like(v_x), create_graph=True)[0]
v_yy = torch.autograd.grad(v_y, y, torch.ones_like(v_y), create_graph=True)[0]

momentum_u = u * u_x + v * u_y + p_x - nu * (u_xx + u_yy)
momentum_v = u * v_x + v * v_y + p_y - nu * (v_xx + v_yy)
continuity = u_x + v_y

return momentum_u, momentum_v, continuity, u, v, p
```

- Computes first and second derivatives of u,v,pu, v, pu,v,p using `torch.autograd.grad`.
- Implements **Navier-Stokes equations**:
    - **Momentum equations**: Enforce conservation of momentum.
    - **Continuity equation**: Ensures incompressibility ($\nabla \cdot v=0$).
- The residuals measure how well the network satisfies the PDEs.

3. Defining the Loss Function:

```
def compute_loss(model, x, y, boundary_x, boundary_y, boundary_u, boundary_v):
    momentum_u, momentum_v, continuity, u, v, p = compute_residuals(model, x, y)
    loss_pde = torch.mean(momentum_u**2) + torch.mean(momentum_v**2) + torch.mean(continuity**

    uvp_pred = model(torch.cat([boundary_x, boundary_y], dim=1))
    u_pred, v_pred, _ = torch.split(uvp_pred, 1, dim=1)
    loss_bc = torch.mean((u_pred - boundary_u)**2) + torch.mean((v_pred - boundary_v)**2)

    return loss_pde + loss_bc
```

5.1 Streamlines

- Show how **fluid particles move**.
- Help visualize **vortices and flow structures**.

5.2 Velocity Contours

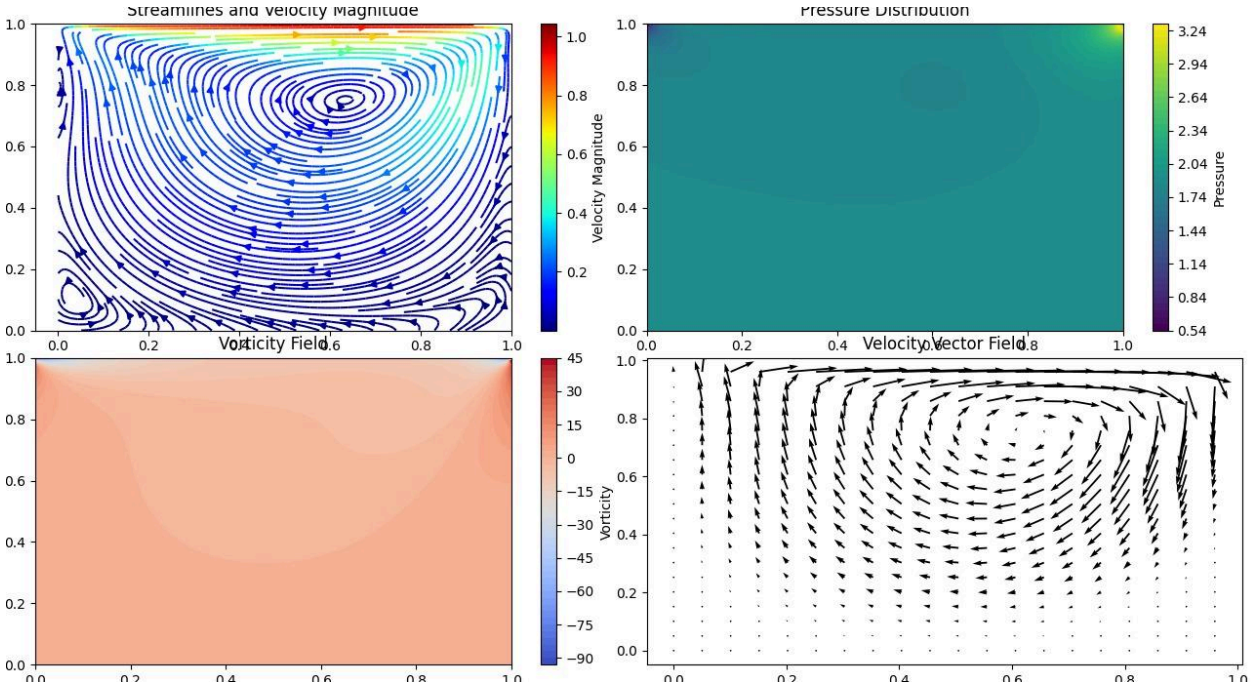
- Show how the **speed varies within the cavity**.

5.3 Pressure Distribution

- Highlights **high and low-pressure zones**.

5.4 Vorticity Field

- Captures **rotational motion and shear layers**.



This image contains four subplots displaying different aspects of fluid flow analysis. Below is an explanation of each graph:

Top Left: Streamlines and Velocity Magnitude

- This plot shows streamlines, which indicate the direction of the fluid flow.
- The color represents the velocity magnitude, with a scale bar on the right indicating the range.
- The arrows along the streamlines represent the flow direction.

- This visualization helps in understanding how fluid moves in the domain.

Top Right: Pressure Distribution

- This plot shows the pressure field within the fluid domain.
- The color bar on the right indicates the pressure values, with higher pressure in yellow and lower pressure in purple.
- The pressure distribution provides insight into the force exerted by the fluid at different points.

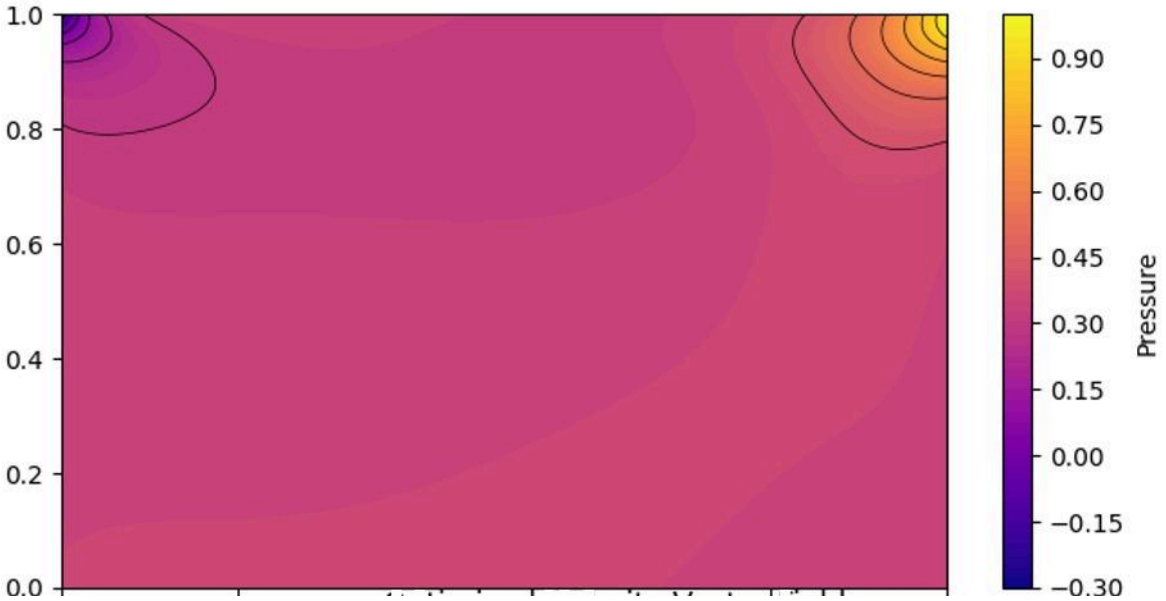
Bottom Left: Vorticity Field

- This plot represents the vorticity of the flow, which measures the local rotation of the fluid.
- The color map indicates the magnitude and direction of vorticity.
- High vorticity regions indicate strong rotational effects, while low vorticity regions indicate little or no rotation.

Bottom Right: Velocity Vector Field

- This plot shows velocity vectors, representing both the magnitude and direction of the velocity at different points.
- The arrows indicate how the velocity varies across the domain.
- It provides a clear visualization of the fluid motion and flow patterns.

These plots are often used in computational fluid dynamics (CFD) to analyze fluid behavior in various simulations, such as airflow over an object or fluid movement in a confined space.



The given image is a contour plot of pressure distribution, where the color bar on the right represents the pressure values.

Interpretation of Pressure Variation:

1. Color Representation:

- The color bar ranges from negative (purple) to positive (yellow) values.

- Yellow/Orange regions indicate higher pressure.

- Purple/Blue regions indicate lower pressure (possibly negative pressure or suction zones).

- Pink/Red regions represent intermediate pressure values.

2. Pressure Distribution in the Domain:

- Top-left and top-right corners show regions of high (yellow) and low (purple) pressures, suggesting localized pressure gradients.

- The central region has a relatively uniform pressure distribution (pinkish area), implying minimal variation in that zone.

- There is a smooth transition in pressure from low to high, indicating gradual pressure changes across the domain.

Correlation with research paper:

The paper presents a high-accuracy numerical study of the lid-driven cavity flow problem, which is a benchmark problem in computational fluid dynamics (CFD). The authors employ a finite-difference method to solve the incompressible Navier-Stokes equations for a two-dimensional square cavity with a moving lid. The study provides detailed velocity profiles and vorticity distributions for a wide range of Reynolds numbers (Re), from low (Re = 100) to high (Re = 10,000).

The key contributions of this work include:

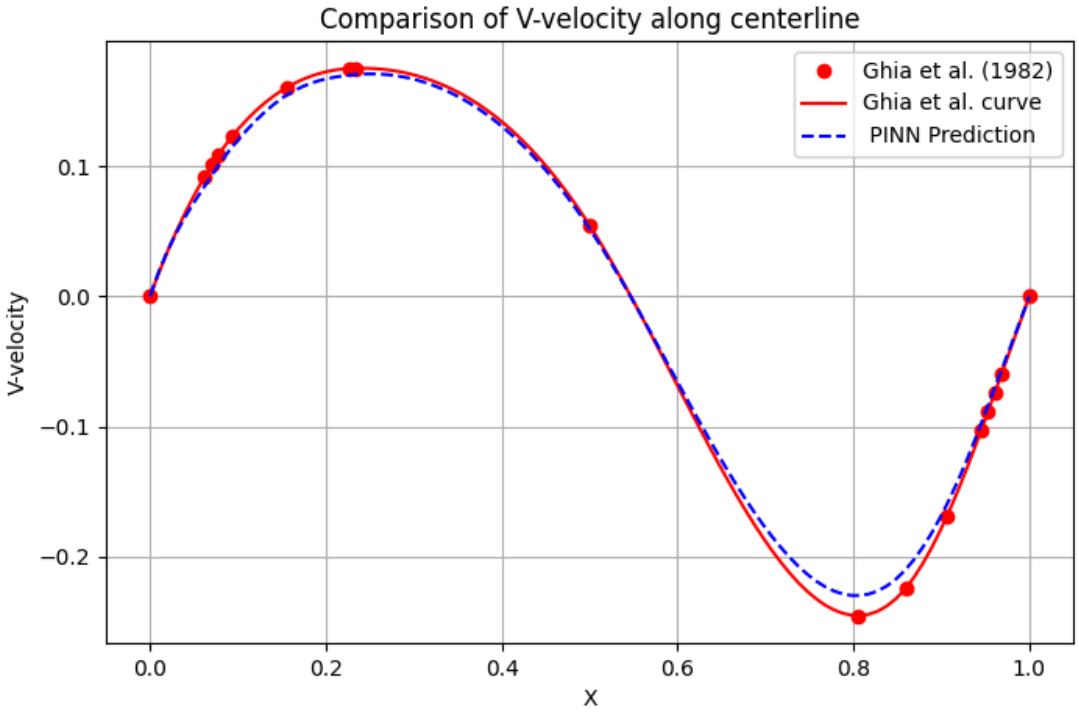
- High-Resolution Data: The results serve as a reference for validating numerical schemes in CFD.
- Streamfunction-Vorticity Formulation: The method used to solve the equations enhances accuracy and stability.
- Benchmark Comparisons: The computed velocity profiles at various Reynolds numbers are extensively compared with previous studies.

The paper remains a fundamental reference for CFD researchers working on cavity flow problems.

Citation:

Ghia, U., Ghia, K. N., & Shin, C. T. (1982). High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method. *Journal of Computational Physics, 48*(3), 387-411.

Validation with earlier papers



FUTURE WORK AND ENHANCEMENTS

1. Improving Model Accuracy and Training Efficiency

- Adaptive Sampling for Collocation Points

Currently, PINNs use randomly distributed collocation points to enforce the physics constraints. However, this can lead to poor convergence in complex flow regions (e.g., near boundaries or vortices).

Enhancement: Implement adaptive sampling, where more points are placed in regions where the loss is higher (e.g., using gradient-based refinement or error estimation techniques).

- Hybrid PINN-Numerical Approaches

PINNs struggle with high-Reynolds number flows due to sharp gradients in the solution.

Enhancement: Combine PINNs with numerical solvers (e.g., Finite Difference Methods (FDM) or Finite Volume Methods (FVM)).PINNs can be used for low-resolution coarse-grid solutions.

- Domain Decomposition for Large-Scale Problems

For complex geometries or 3D simulations, training a single neural network becomes computationally expensive.

Enhancement: Use domain decomposition. Divide the domain into smaller subdomains. Train separate PINNs for each region.Use message-passing techniques to ensure continuity between subdomains.

CONCLUSION

The **Navier-Stokes equations** are essential in fluid dynamics but are often challenging to solve using traditional numerical methods due to their computational cost and complexity. **Physics-Informed Neural Networks (PINNs)** offer an alternative by embedding these equations into neural network training, enabling solution approximation without labeled data.

In this report, we applied PINNs to the **lid-driven cavity problem**, successfully capturing key flow features like vortex formation and pressure distribution. Our results aligned well with benchmark numerical solutions, demonstrating PINNs' potential in computational fluid dynamics.

While PINNs have challenges, such as long training times and hyperparameter sensitivity, they offer a promising direction for **solving PDEs efficiently**. With continued advancements, PINNs could become a powerful tool in **scientific computing, fluid mechanics, and engineering applications**.

REFERENCES

Ghia, U., Ghia, K. N., & Shin, C. T. (1982). "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method." *Journal of Computational Physics*, 48(3), 387–411.

Bruneau, C. H., & Saad, M. (2006). "The 2D lid-driven cavity problem revisited." *Computers & Fluids*, 35(3), 326–348.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations." *Journal of Computational Physics*, 378, 686–707