Physics-Informed Neural Networks: Navier-Stokes Equations

Olivia Bouvier
Matthew Goldstein
Saimouli Katragadda

Table of contents

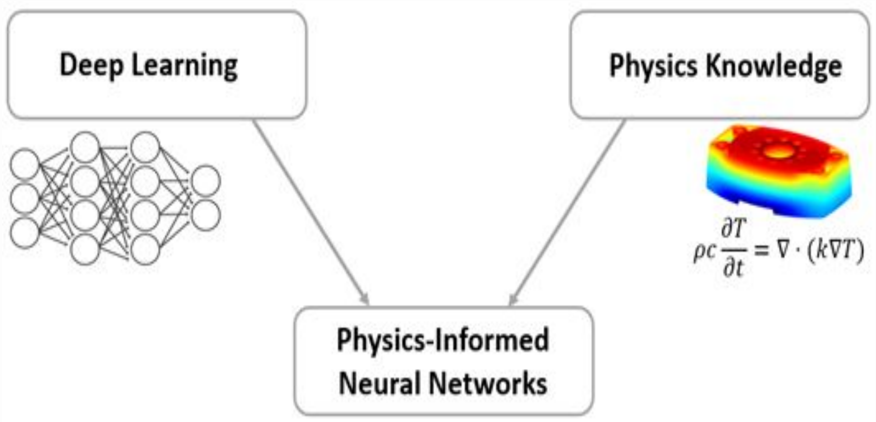
Introduction

What are PINNs?

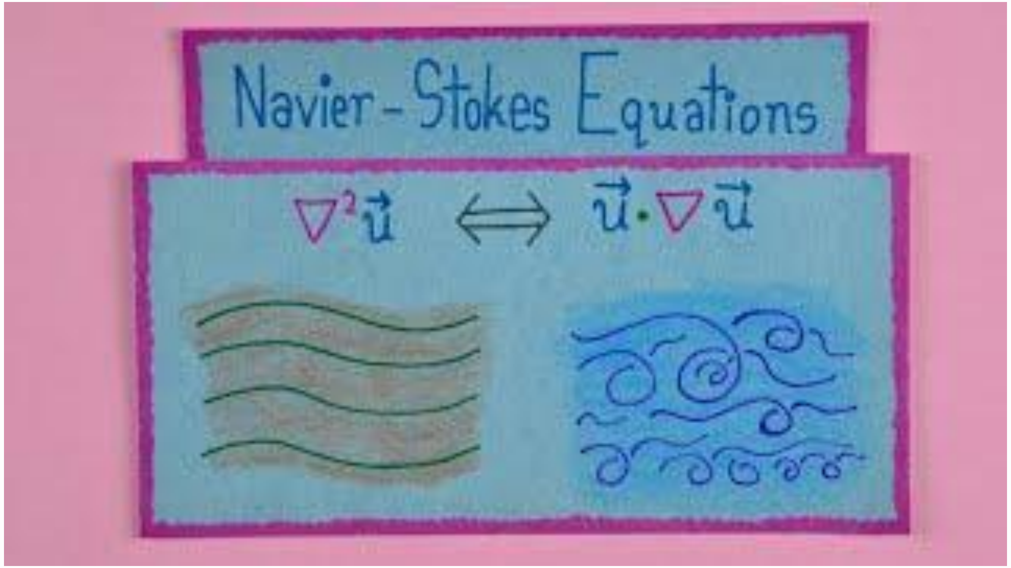
- Powerful type of neural network
- Work by embedding physics into the loss function
- Often use PDEs or ODEs to represent physics

Why are they useful?

- Mesh-free
- Leverage known physics to get results using limited data
- Require fewer data points than traditional models
- Can handle noisy data efficiently



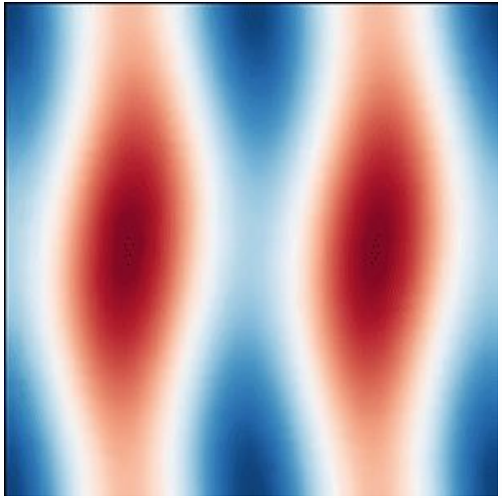
The Navier-Stokes Equation



Motivation

Why Navier-Stokes in Bioinformatics?

- Fluid mechanics is one of the most widely applicable branches of classic continuum physics
- Can be used to model blood flow and circulation, as well as cerebrospinal fluid, air in the lungs, and other organ-level simulations
- The 2D incompressible Navier-Stokes Equations capture the physics of fluid motion, including viscosity and pressure



Why use PINNs for Navier-Stokes?

- Classical numerical methods require the generation of high-quality meshes
- Mesh generation consumes a significant portion of the total simulation pipeline
- Mesh fine-tuning leads to steep increases in CPU time and memory usage
- PINNS avoid this by embedding the governing equations directly into the loss function
- They employ collocation points in the physical domain and penalize the PDE residuals
- Can naturally handle irregular domains, moving boundaries, and multi-physics coupling

Methodology

Overview

- **Objective**: Learn the solution to 2D incompressible Navier–Stokes equations and simultaneously infer physical parameters

Approach:

- Stream function formulation ψ
- Construct a physics-informed loss that includes both data and PDE residuals.
- Two-stage training: Adam + L-BFGS

Governing Equations

$$\frac{\partial u}{\partial t} + \boxed{\lambda_1}\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial p}{\partial x} + \boxed{\lambda_2}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$
$$\frac{\partial v}{\partial t} + \boxed{\lambda_1}\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = -\frac{\partial p}{\partial y} + \boxed{\lambda_2}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right),$$
$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0.$$

- Unknowns: *u, v, p (velocities, pressure)*
- Parameters to infer: λ_1(convection), λ_2 (diffusion)

Can be written as:

$$f_u = u_t + \lambda_1(uu_x + vu_y) + p_x - \lambda_2(u_{xx} + u_{yy})$$
$$f_v = v_t + \lambda_1(uv_x + vv_y) + p_y - \lambda_2(v_{xx} + v_{yy})$$
= 0

Stream Function Representation

$$u = \frac{\partial \psi}{\partial y},$$

$$v = -\frac{\partial \psi}{\partial x}.$$

- To enforce incompressibility, we represent the velocity components in terms of a scalar stream function $\psi\,(x, y, t)$
- This automatically satisfies the incompressibility constraint $\nabla \cdot u = 0$ (implicit)

PINN Architecture

- Inputs: (x, y, t)
- Outputs: (ψ, p)
- Architecture:
    - 9 hidden layers, 20 neurons
- Velocity obtained via auto-diff of ψ

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}$$

Loss Components

Total loss

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{physics}}$$

Data loss:

$$\mathcal{L}_{\text{data}} = \text{MSE}(u_{\text{pred}}, u_{\text{true}}) + \text{MSE}(v_{\text{pred}}, v_{\text{true}})$$

Physics loss:

$$\mathcal{L}_{\text{physics}} = \text{MSE}(f_u) + \text{MSE}(f_v)$$

Training

- Normalize all coordinates to [-1, 1] for stability

- During a forward pass, given inputs (x, y, t), the network returns predictions for ψ and p , use automatic diff. $u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}$

- These are then substituted into the Navier–Stokes equations to compute residuals

Optimization Strategy

- Stage 1 – Adam Optimizer
- Adaptive, first-order optimizer
- Fast initial convergence

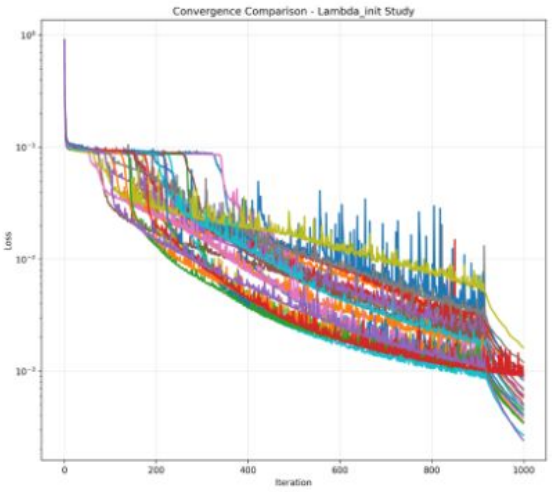
- Stage 2 – Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) Optimizer
- Quasi-Newton method
- Refines weights and physical parameters using curvature info

Experiments

Robustness to Initialization

- Explored how different initial values for the unknown parameters λ_1 and λ_2 affect the convergence, accuracy, and stability
- The PINN recovered accurate values of λ_1 and λ_2 from a range of initializations
- Initialization had little effect on velocity prediction accuracy (u, v)
- Could introduce large deviations in pressure and parameter errors, especially for higher λ_1 initial values

Init (λ_1, λ_2)	e_u	e_v	e_p	e_{λ_1}	e_{λ_2}	Loss	Smooth
(0.0, 0.00)	0.0116	0.0315	2.94	0.537	8.32	0.00046	0.0192
(0.0, 0.01)	0.0081	0.0294	2.25	0.309	8.10	0.00035	0.0192
(0.5, 0.05)	0.0078	0.0254	8.76	0.342	7.36	0.00027	0.0192
(2.0, 0.02)	0.0235	0.0704	2.78	1.466	15.67	0.00162	0.0193
(5.0, 0.05)	0.0078	0.0239	8.20	0.279	5.46	0.00024	0.0193

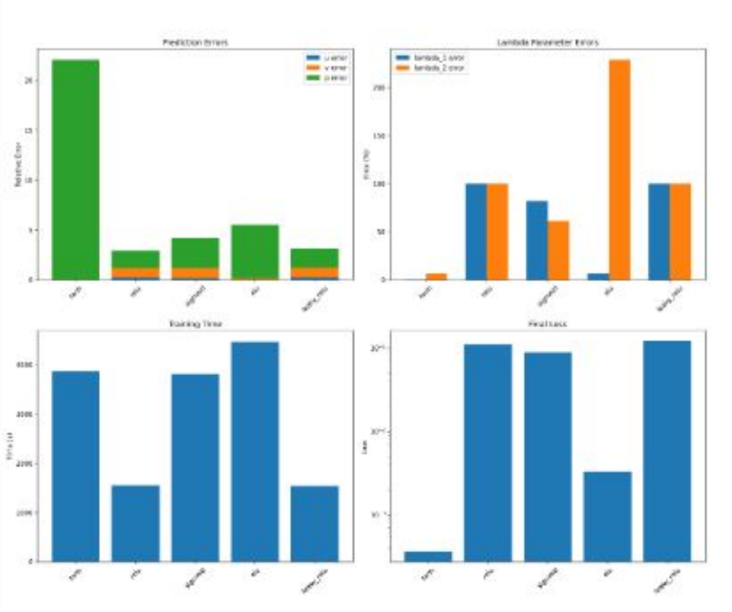


Robustness to Noise

- We conducted ablation experiments by injecting additive Gaussian noise into the input data at five levels: 0.0, 0.01, 0.05, 0.1, and 0.2
- Used the tanh activation function and Adam optimizer
- PINN exhibited stable performance in velocity prediction across all noise levels, with only minor error increases
- Pressure predictions were more sensitive to noise
- The viscosity parameter was also sensitive to noise
- Training time and loss did not correlate linearly with noise level, illustrating the optimizer's robustness and the value of the two-stage training strategy

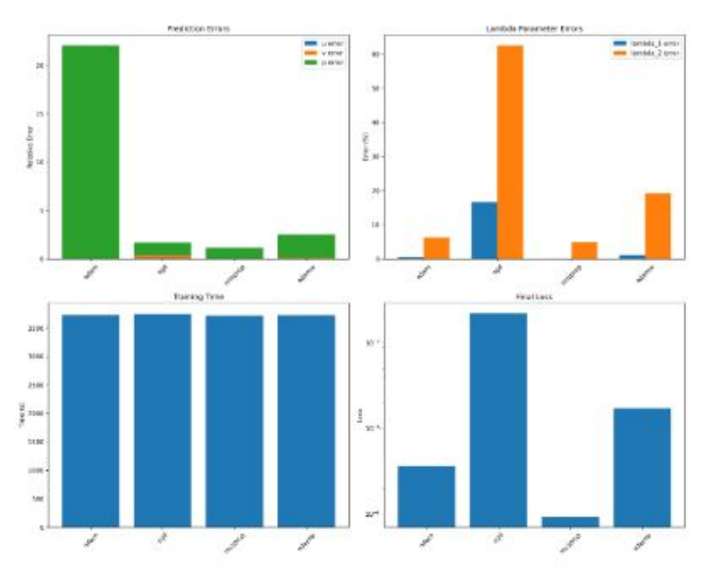
Noise	e_u	e_v	e_p	e_{λ_1}	e_{λ_2}	Loss	Time (s)	Smooth
0.00	0.0119	0.0317	3.033	0.496	8.335	0.00047	4305	0.0192
0.01	0.0177	0.0443	28.640	0.735	10.590	0.00079	4444	0.0192
0.05	0.0132	0.0364	1.448	0.407	6.954	0.00072	4945	0.0191
0.10	0.0145	0.0434	2.575	0.824	7.790	0.00212	4620	0.0193
0.20	0.0148	0.0388	2.809	0.519	4.996	0.00622	4563	0.0195

Activation Study



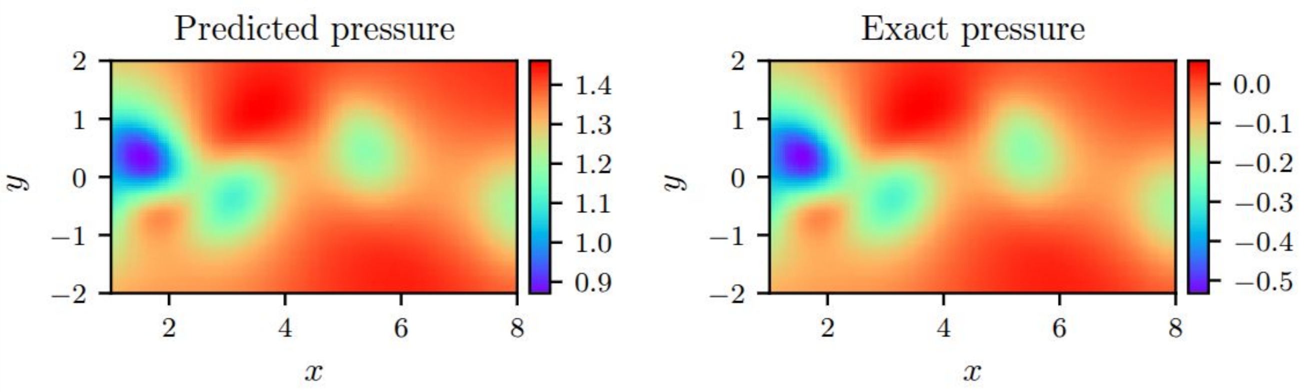
- Tested 5 different activation functions: tanh, ReLU, sigmoid eLU, and leaky ReLU
- ReLU and leaky ReLU were the fastest as shown in the figure
- Tanh was the best option with the lowest loss and lambda errors

Optimizer Study



- Tested Adam, Stochastic Gradient Descent (SGD), MSprop, and AdamW optimizers
- No noticeable difference in training time
- MSProp had the least final loss

Takeaways



- Moderate initial values near expected physical ranges offer the best trade-off between stability and accuracy
- Tanh had the best results of the activation function
- MSProp was the best optimizer for our PINN

Future Work

- The experiments reveal several challenges and limitations inherent to the PINN methodology
- Sensitivity to noise
- More effective activation functions led to longer computing time
- Extending this framework to larger-scale or three-dimensional flows will require addressing scalability challenges
- Physics-informed transformers might improve expressiveness and generalization to more complex or chaotic flow regimes

References

- 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 (2019), 686–707.
- Jens Berg and Kaj Nyström. 2018. A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 317 (2018), 28–41.
- Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. 2020. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive Physics Informed Neural Networks - Navier Stokes 4D flow MRI data using physics-informed neural networks. Computer Methods in Applied Mechanics and Engineering 358 (2020), 112623.
- Justin Sirignano and Konstantinos Spiliopoulos. 2018. DGM: A deep learning algorithm for solving partial differential equations. Journal of computational physics 375 (2018), 1339–1364.

Q/A