Deep learning techniques for Blood Flow simulation based on Navier-Stokes Equations

Alejandro Salazar Arango[1]

Advisor(s):
Cristhian David Montoya Zambrano[2]

Research practice 3
Research proposal
Mathematical Engineering
School of Applied Sciences and Engineering
Universidad EAFIT

August 2023

[1]Mathematical Engineering Student. Universidad EAFIT asalazara1@eafit.edu.co
[2]School of Applied Sciences and Engineering. Universidad EAFIT cdmontoyaz@eafit.edu.co

Abstract

The purpose of this paper is to
present a simulation of blood flow through the carotid artery in two dimensions. The simulation is performed on a section of the artery extracted from an angiography, on which modifications are made to mimic the presence of stenosis in the artery. Finite element method is used to solve the linear simplification of the Navier-Stokes equations, seeking to identify variations in the pressure and velocity fields of the fluid and to analyze the impact of mild and severe cases of stenosis caused by atherosclerosis on human hemodynamics.

Keywords: Finite element method, Computational fluid dynamics, Hemodynamics, Ischemia, Stroke

1 Introduction

Partial differential equations (PDEs) are one of the most powerful tools for the study of physical and biological phenomena. They allow us to represent in a simplified way, the dynamics between the spatial and temporal variables of any phenomenon and to take these dynamics to a mathematical model, whose solution describes in a great way the behavior of the studied system[1]. This has led partial differential equations to become a tool of great use in areas such as fluid dynamics, electromagnetism, optics, magnetism, among others[2]; these areas being dominated by PDEs such as Maxwell's Equations and Navier-Stokes Equations.

However, PDEs, given their complex structure, are usually models whose analytical solutions are difficult or impossible to find, being even the solutions already found for some PDEs so complex that it is preferred to use other methods to solve the problem[3]. This is why, in recent years, the development of multiple numerical methods to solve this type of equations has been one of the major topics of study of the scientific community. In addition, numerical methods for solving PDEs are a really important topic of study given that although there are already multiple methods developed, if these are not applied correctly can lead to solutions far from the real solution of the problem and therefore lead to erroneous conclusions about the behavior of the system under study.

One of the most commonly used numerical method for solving Partial Differential Equations is the Finite Element Method (FEM), a numerical method consisting on the discretization of the problem's domain, to later solve a variational formulation of the problem with determined test functions, to obtain an algebraic system of equations, which, when solved, gives the approximated solution of the problem. However, due to the substantial number of degrees of freedom it needs in order to correctly solve the PDE, it remains as an extremely expensive method both in terms of CPU and memory demand, therefore not being too useful for real-time contexts[4] [5]. Consequently, models such as physically informed neural networks (PINNs) have been developed and have become important methods in the study of

PDEs.

PINNs are quite useful in this type of problems since, by incorporating the physical information of the system under study in the training of the neural networks, allowing the imposition of boundary conditions and governing equations directly on the network architecture, it facilitates the adaptation of the network to different conditions and geometries on which to solve the problem, in addition to potentially even improving the accuracy of the numerical solutions.

Therefore thi project focuses on the implementation of multiple Physical Informed Neural Networks (PINNs) architechtures for solving Fluid Dynamic problems, seeking to observe and compare the performance of these architectures, aiming for a deeper understanding of the advantages and limitations of PINNs in the context of Fluid dynamics simulations.

2 State of the art

Given its important on a great amount of biological, medical, and physical fields, PDEs are one of the most studied topics in applied mathematics, and therefore, a large body of literature on the subject can be found. The following is a review of some of the literature consulted on PINNs and RBMs and their applications in EDPs, where a collection of both theoretical and applied papers are presented.

On the other hand, although PINNs are new techniques in the solution of PDEs, they have also been shown to be a topic of great interest to the scientific community and therefore it is not difficult to find literature on the subject, not only for forward problems but for inverse problems as well. On this matter, Berg & Nyström [6] , Meng & Karniadakis [7] and Raissi, Perdikaris & Karniadakis [8] present three works of interest in which they use PINNs as a solution method for inverse problems with multiple examples and making a focus on the explanation of how PINNs work on every presented examples, as well as a mathematical background on the method itself.

Some works about the usage of PINN for fluid dynamics problems can be found as well. A first interesting work on this topic can be found in the paper from Moschou *et al.* [9] who present an application of PINNs on the modeling of astrophysical shocks, focusing on the solar termination shock, a phenomenon described using the fluid dynamics conservation laws. In agreement Ciao *et al.* [10] developed an application of PINN for the turbulent mixing induced by the Rayleigh-Taylor instability, using the Reynolds-Averaged Navier–Stokes Equations for modeling the problem. And finally, Lino *et al.* [11] exhibit a review of the current and emerging deep-learning methods used on fluid dynamics simulation.

3 Proposed methodology

As a means to achieve the objectives set for this project, a 3-stage methodology was proposed in which various mathematical methods were used. In the first stage a version of the Broyden–Fletcher–Goldfarb–Shanno algorithm, the L-BFGS-B optimization method, was implemented, which is the optimization method used for the training of the PINNs implemented. The second stage consisted on the implementation, training and validation of multiple PINN achitectures, which were used for solving both Burgers and Navier Stokes equations with a variety of cases for comparing the capacity of these Neural Networks for solving these problems. Finally the third stage was the aalysis of results and the comparison of the performances of the models implemented on the previous stage.

Below both L-BFGS-B an PINN are explained with more detail for the reader to understand the nature of both algorithms and how they work.

3.1 The Broyden–Fletcher–Goldfarb–Shanno algorithm

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is an iterative method for numerical optimization, specifically used for finding the minimum of a function. It falls under the category of quasi-Newton methods and is particularly well-suited for problems where the objective function is smooth and has continuous derivatives. The main goal of the BFGS algorithm is to iteratively update an approximation of the inverse Hessian matrix, which represents the local curvature of the objective function, to efficiently converge towards the minimum.

At each iteration, the BFGS algorithm computes the gradient of the objective function with respect to the parameters, in order to subsequently update the current estimate of the inverse Hessian matrix based on the difference in gradients and parameter updates between iterations as shown in Equation 1. By utilizing this information, the algorithm generates a sequence of parameter values that progressively move closer to the optimal solution. Unlike Newton's method, BFGS avoids explicitly computing the Hessian matrix, which can be computationally expensive for large problems, by iteratively updating an approximation of it [12].

$$\begin{aligned} s_k &= x_{k+1} + x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \\ H_{k+1}^{-1} &= (I - \frac{sy^T}{y^T s}) H_k^{-1} \frac{ys^T}{y^T s} + \frac{ss^T}{y^T s} \end{aligned} \tag{1}$$

BFGS' use of an approximation of the inverse Hessian matrix allows for a balance between efficiency and accuracy, making it a popular choice for optimizing smooth and high-dimensional functions. However, BFGS does not guarantee convergence to a global minimum and may

converge to a local minimum based on the starting point and the characteristics of the objective function.

3.2 Physical Informed neural Networks

Physics-informed neural networks (PINNs) have emerged as a promising approach to solve PDEs with remarkable efficiency and accuracy. These combine the power of neural networks with the governing equations of physical systems, offering a seamless integration of data-driven learning and domain-specific physics. They work by incorporating the residual error of the governing equations into the neural network training process, gaining a unique ability to capture the underlying physics of the model while learning from sparse and noisy data[8].

Unlike traditional methods, PINNs avoid the need for explicit mesh discretization and provide continuous solutions throughout the domain, dynamically adapting to complex geometries. Their versatility, combined with the ability to leverage existing deep learning frameworks, positions PINNs as a transformative tool for advancing EDP-based modeling and simulation in interdisciplinary domains.

4 Results

For this project three set of equations were considered with multiple scenarios and architechtures each. The main results obtained for each set of equations is presented below

4.1 1-dimentional Burgers' Equation

For the 1-dimentional Burgers0 Equation, three cases were implemeneted each with the listed three architectures:

- Architechture 1: 5 hidden layers with 20 neurons each
- Architechture 2: 5 hidden layers with 50 neurons each
- Architechture 3: 8 hidden layers with 20 neurons each

Case 1:

For the first case the following exact solution $u(x,t)$ was considered

$$u(x,t) = e^{-t}\sin(\pi x)$$

which, for $\nu = \frac{1}{\pi^2}$, gives the PDE below

$$u_t + uu_x - \frac{1}{\pi^2} u_{xx} = \pi e^{-2t} \sin(\pi x) \cos(\pi x) \tag{2}$$

When solving Equation 2 with the architectures explained above, the following results were obtained

Time	**MSE**	l^2 **error**
0.0	4.1341×10^{-8}	2.0332×10^{-3}
0.1	1.7436×10^{-8}	1.3205×10^{-3}
0.2	9.8793×10^{-9}	9.9395×10^{-4}
0.3	7.5820×10^{-9}	8.7074×10^{-4}
0.4	5.1086×10^{-9}	7.1475×10^{-4}
0.5	2.1645×10^{-9}	4.6525×10^{-4}
0.6	1.1935×10^{-9}	3.4546×10^{-4}
0.7	2.6267×10^{-9}	5.1251×10^{-4}
0.8	2.9190×10^{-9}	5.4027×10^{-4}
0.9	8.5705×10^{-10}	2.9275×10^{-4}
1.0	1.0075×10^{-8}	1.0037×10^{-3}

Table 1: Results for the first architecture in the first case of Burgers Equations

Time	**MSE**	l^2 **error**
0.0	4.8725×10^{-9}	6.9803×10^{-4}
0.1	8.4316×10^{-9}	9.1824×10^{-4}
0.2	1.3853×10^{-8}	1.1770×10^{-3}
0.3	1.4626×10^{-8}	1.2094×10^{-3}
0.4	1.0354×10^{-8}	1.0176×10^{-3}
0.5	5.4203×10^{-9}	7.3623×10^{-4}
0.6	4.0029×10^{-9}	6.3269×10^{-4}
0.7	5.2350×10^{-9}	7.2353×10^{-4}
0.8	4.1608×10^{-9}	6.4504×10^{-4}
0.9	1.2808×10^{-9}	3.5789×10^{-4}
1.0	2.4014×10^{-8}	1.5497×10^{-3}

Table 2: Results for the second architecture in the first case of Burgers Equations

Time	**MSE**	l^2 **error**
0.0	2.7134×10^{-9}	5.2090×10^{-4}
0.1	1.2367×10^{-9}	3.5167×10^{-4}
0.2	4.1095×10^{-9}	6.4105×10^{-4}
0.3	6.9692×10^{-9}	8.3482×10^{-4}
0.4	6.4658×10^{-9}	8.0410×10^{-4}
0.5	4.1114×10^{-9}	6.4120×10^{-4}
0.6	2.0376×10^{-9}	4.5139×10^{-4}
0.7	8.9270×10^{-10}	2.9878×10^{-4}
0.8	4.2240×10^{-10}	2.0552×10^{-4}
0.9	3.2816×10^{-10}	1.8115×10^{-4}
1.0	1.0638×10^{-9}	3.2616×10^{-4}

Table 3: Results for the third architecture in the first case of Burgers Equations

Case 2:

For the second case the following exact solution $u(x,t)$ was considered

$$u(x,t) = e^t(x - x^2)$$

which, for $\nu = 1$, gives the PDE below

$$u_t + uu_x - u_{xx} = e^t(x - x^2) + e^{2t}(x + x^2)(1 - 2x) + 2e^t \tag{3}$$

When solving Equation 3 with the architectures explained above, the following results were obtained

Time	**MSE**	l^2 **error**
0.0	2.7535×10^{-1}	5.2474
0.1	1.9203×10^{-1}	4.3821
0.2	1.2686×10^{-1}	3.5618
0.3	7.7156×10^{-2}	2.7777
0.4	4.0875×10^{-2}	2.0218
0.5	1.6563×10^{-2}	1.2870
0.6	3.2628×10^{-3}	0.5712
0.7	4.6056×10^{-4}	0.2146
0.8	8.0537×10^{-3}	0.8974
0.9	2.6332×10^{-2}	1.6227
1.0	5.5973×10^{-2}	2.3659

Table 4: Results for the first architecture in the second case of Burgers Equations

Time	**MSE**	l^2 **error**
0.0	2.7545×10^{-1}	5.2483
0.1	1.9191×10^{-1}	4.3807
0.2	1.2675×10^{-1}	3.5602
0.3	7.7123×10^{-2}	2.7771
0.4	4.0908×10^{-2}	2.0226
0.5	1.6612×10^{-2}	1.2889
0.6	3.2859×10^{-3}	0.5732
0.7	4.5513×10^{-4}	0.2133
0.8	8.0546×10^{-3}	0.8975
0.9	2.6372×10^{-2}	1.6239
1.0	5.5989×10^{-2}	2.3662

Table 5: Results for the second architecture in the second case of Burgers Equations

Time	**MSE**	l^2 **error**
0.0	2.7520×10^{-1}	5.2459
0.1	1.9223×10^{-1}	4.3844
0.2	1.2703×10^{-1}	3.5641
0.3	7.7172×10^{-2}	2.7780
0.4	4.0783×10^{-2}	2.0195
0.5	1.6462×10^{-2}	1.2830
0.6	3.2196×10^{-3}	0.5674
0.7	4.6870×10^{-4}	0.2165
0.8	8.0401×10^{-3}	0.8967
0.9	2.6244×10^{-2}	1.6200
1.0	5.5965×10^{-2}	2.3657

Table 6: Results for the third architecture in the second case of Burgers Equations

Case 3:

For the third case the following exact solution $u(x,t)$ was considered

$$u(x,t) = te^{x-x^2}$$

which, for $\nu = 1$, gives the PDE below

$$u_t + uu_x - u_{xx} = e^{x-x^2} + t^2 e^{2(x-x^2)}(1-2x) - te^{x-x^2}(4x^2 - 4x - 1) \tag{4}$$

When solving Equation 4 with the architectures explained above, the following results were obtained

Time	**MSE**	l^2 **error**

Table 7: Results for the first architecture in the third case of Burgers Equations

Time	**MSE**	l^2 **error**

Table 8: Results for the second architecture in the third case of Burgers Equations

Time	MSE	l^2 error
0.0	3.9095×10^{-9}	6.2526×10^{-4}
0.1	4.0976×10^{-9}	6.4012×10^{-4}
0.2	1.2145×10^{-8}	1.1020×10^{-3}
0.3	1.9778×10^{-8}	1.4063×10^{-3}
0.4	2.1008×10^{-8}	1.4494×10^{-3}
0.5	1.3044×10^{-8}	1.1421×10^{-3}
0.6	3.4472×10^{-9}	5.8713×10^{-4}
0.7	1.7965×10^{-9}	4.2385×10^{-4}
0.8	6.2565×10^{-9}	7.9098×10^{-4}
0.9	4.9122×10^{-9}	7.0087×10^{-4}
1.0	1.9582×10^{-8}	1.3994×10^{-3}

Table 9: Results for the third architecture in the third case of Burgers Equations

4.2 1-dimentional Navier-Stokes Equations

For the 1-dimentional Navier-Stokes Equations, two cases were implemeneted each with the listed four architectures:

- Architechture 1: 5 hidden layers with 20 neurons each
- Architechture 2: 5 hidden layers with 50 neurons each
- Architechture 3: 8 hidden layers with 20 neurons each
- Architechture 4: 8 hidden layers with 50 neurons each

Case 1:

For the first case the following exact solution $u(x,t)$ and $p(x,t)$ was considered

$$u(x,t) = e^{-t}\sin(\pi x)$$
$$p(x,t) = -\sin(\pi x)$$

which, for $\nu = \frac{1}{\pi^2}$, gives the PDE below

$$u_t + uu_x - \frac{1}{\pi^2}u_{xx} + p_x = \pi e^{-2t}\left(\sin(\pi x)\cos(\pi x) - \cos(\pi x)\right) \tag{5}$$

When solving Equation 5 with the architectures explained above, the following results were obtained

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	1.9646×10^{-7}	0.0044	9.8202×10^{-6}	0.0313
0.1	6.8350×10^{-7}	0.0083	1.2453×10^{-5}	0.0353
0.2	1.7235×10^{-6}	0.0131	1.6410×10^{-5}	0.0405
0.3	3.2262×10^{-6}	0.0180	2.1246×10^{-5}	0.0461
0.4	5.4656×10^{-6}	0.0234	2.7291×10^{-5}	0.0522
0.5	8.6866×10^{-6}	0.0295	3.4859×10^{-5}	0.0590
0.6	1.3048×10^{-5}	0.0361	4.3914×10^{-5}	0.0663
0.7	1.8639×10^{-5}	0.0432	5.3911×10^{-5}	0.0734
0.8	2.5493×10^{-5}	0.0505	6.3784×10^{-5}	0.0799
0.9	3.3628×10^{-5}	0.0580	7.2126×10^{-5}	0.0849
1.0	4.3075×10^{-5}	0.0656	7.7515×10^{-5}	0.0880

Table 10: Results for the first architecture in the first case of the 1-dimensional Navier-Stokes Equations

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	9.8856×10^{-9}	0.0010	2.4502×10^{-6}	0.0157
0.1	2.4990×10^{-7}	0.0050	2.4867×10^{-6}	0.0158
0.2	7.4739×10^{-7}	0.0086	3.3587×10^{-6}	0.0183
0.3	1.4044×10^{-6}	0.0119	5.0945×10^{-6}	0.0226
0.4	2.2494×10^{-6}	0.0150	7.7558×10^{-6}	0.0278
0.5	3.3342×10^{-6}	0.0183	1.1329×10^{-5}	0.0337
0.6	4.7053×10^{-6}	0.0217	1.5696×10^{-5}	0.0396
0.7	6.3922×10^{-6}	0.0253	2.0645×10^{-5}	0.0454
0.8	8.3984×10^{-6}	0.0290	2.5889×10^{-5}	0.0509
0.9	1.0697×10^{-5}	0.0327	3.1097×10^{-5}	0.0558
1.0	1.3233×10^{-5}	0.0364	3.5934×10^{-5}	0.0599

Table 11: Results for the second architecture in the first case of the 1-dimensional Navier-Stokes Equations

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	6.1127×10^{-7}	0.0078	1.1934×10^{-4}	0.1092
0.1	1.0509×10^{-5}	0.0324	2.1964×10^{-4}	0.1482
0.2	3.9397×10^{-5}	0.0628	3.3119×10^{-4}	0.1820
0.3	8.3767×10^{-5}	0.0915	4.4435×10^{-4}	0.2108
0.4	1.3836×10^{-4}	0.1176	5.5427×10^{-4}	0.2354
0.5	1.9779×10^{-4}	0.1406	6.5851×10^{-4}	0.2566
0.6	2.5804×10^{-4}	0.1606	7.5561×10^{-4}	0.2749
0.7	3.1720×10^{-4}	0.1781	8.4431×10^{-4}	0.2906
0.8	3.7555×10^{-4}	0.1938	9.2328×10^{-4}	0.3039
0.9	4.3517×10^{-4}	0.2086	9.9122×10^{-4}	0.3148
1.0	4.9946×10^{-4}	0.2235	1.0470×10^{-3}	0.3236

Table 12: Results for the third architecture in the first case of the 1-dimensional Navier-Stokes Equations

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	1.1785×10^{-7}	0.0034	1.1033×10^{-5}	0.0332
0.1	3.4148×10^{-7}	0.0058	7.1700×10^{-6}	0.0268
0.2	9.7519×10^{-7}	0.0099	5.3050×10^{-6}	0.0230
0.3	2.1471×10^{-6}	0.0147	5.0460×10^{-6}	0.0225
0.4	4.0106×10^{-6}	0.0200	6.1678×10^{-6}	0.0248
0.5	6.6570×10^{-6}	0.0258	8.5383×10^{-6}	0.0292
0.6	1.0104×10^{-5}	0.0318	1.2097×10^{-5}	0.0348
0.7	1.4311×10^{-5}	0.0378	1.6868×10^{-5}	0.0411
0.8	1.9196×10^{-5}	0.0438	2.2987×10^{-5}	0.0479
0.9	2.4648×10^{-5}	0.0496	3.0768×10^{-5}	0.0555
1.0	3.0525×10^{-5}	0.0552	4.0788×10^{-5}	0.0639

Table 13: Results for the fourth architecture in the first case of the 1-dimensional Navier-Stokes Equations

Case 2:

For the first case the following exact solution $u(x,t)$ and $p(x,t)$ was considered

$$u(x,t) = e^t(x - x^2)$$
$$p(x,t) = -\sin(\pi x)$$

which, for $\nu = 1$, gives the PDE below

$$u_t + uu_x - u_{xx} + p_x = e^t(x - x^2) + e^{2t}(x + x^2)(1 - 2x) + 2e^t - \pi \cos(\pi x) \tag{6}$$

When solving Equation 6 with the architectures explained above, the following results were obtained

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**

Table 14: Results for the first architecture in the second case of the 1-dimensional Navier-Stokes Equations

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	8.6068×10^{-7}	0.0093	2.8796×10^{-3}	0.5366
0.1	2.4480×10^{-6}	0.0156	3.2165×10^{-3}	0.5671
0.2	3.6892×10^{-6}	0.0192	3.7059×10^{-3}	0.6088
0.3	4.4041×10^{-6}	0.0210	4.3900×10^{-3}	0.6626
0.4	5.3148×10^{-6}	0.0231	5.3268×10^{-3}	0.7298
0.5	6.9956×10^{-6}	0.0264	6.5958×10^{-3}	0.8121
0.6	9.9863×10^{-6}	0.0316	8.3072×10^{-3}	0.9114
0.7	1.5001×10^{-5}	0.0387	1.0615×10^{-2}	1.0303
0.8	2.2718×10^{-5}	0.0477	1.3734×10^{-2}	1.1719
0.9	3.3336×10^{-5}	0.0577	1.7967×10^{-2}	1.3404
1.0	4.6835×10^{-5}	0.0684	2.3737×10^{-2}	1.5407

Table 15: Results for the second architecture in the second case of the 1-dimensional Navier-Stokes Equations

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	1.7844×10^{-6}	0.0134	5.9847×10^{-3}	0.7736
0.1	2.4234×10^{-5}	0.0492	2.0570×10^{-2}	1.4342
0.2	1.0285×10^{-4}	0.1014	4.6340×10^{-2}	2.1527
0.3	2.6243×10^{-4}	0.1620	8.6240×10^{-2}	2.9367
0.4	5.3076×10^{-4}	0.2304	1.4432×10^{-1}	3.7990
0.5	9.3961×10^{-4}	0.3065	2.2584×10^{-1}	4.7522
0.6	1.5272×10^{-3}	0.3908	3.3733×10^{-1}	5.8080
0.7	2.3419×10^{-3}	0.4839	4.8673×10^{-1}	6.9766
0.8	3.4460×10^{-3}	0.5870	6.8356×10^{-1}	8.2678
0.9	4.9188×10^{-3}	0.7013	9.3924×10^{-1}	9.6914
1.0	6.8590×10^{-3}	0.8282	1.2677	11.2590

Table 16: Results for the third architecture in the second case of the 1-dimensional Navier-Stokes Equations

	$u(x,t)$		$p(x,t)$	
Time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	2.4028×10^{-6}	0.0155	6.6235×10^{-3}	0.8139
0.1	3.3048×10^{-5}	0.0575	2.2045×10^{-2}	1.4848
0.2	1.3703×10^{-4}	0.1171	5.0095×10^{-2}	2.2382
0.3	3.4117×10^{-4}	0.1847	9.4570×10^{-2}	3.0752
0.4	6.7964×10^{-4}	0.2607	1.6022×10^{-1}	4.0027
0.5	1.1959×10^{-3}	0.3458	2.5301×10^{-1}	5.0300
0.6	1.9448×10^{-3}	0.4410	3.8045×10^{-1}	6.1681
0.7	2.9953×10^{-3}	0.5473	5.5208×10^{-1}	7.4302
0.8	4.4333×10^{-3}	0.6658	7.8000×10^{-1}	8.8318
0.9	6.3649×10^{-3}	0.7978	1.0797	10.3908
1.0	8.9200×10^{-3}	0.9445	1.4709	12.1279

Table 17: Results for the fourth architecture in the first case of the 1-dimensional Navier-Stokes Equations

4.3 2-dimentional Navier-Stokes Equations

For the 2-dimentional Navier-Stokes Equations, only one case was implemeneted with the listed five architectures:

- Architechture 1: 5 hidden layers with 50 neurons each
- Architechture 2: 8 hidden layers with 20 neurons each

- Architechture 3: 8 hidden layers with 50 neurons each
- Architechture 3: 8 hidden layers with 70 neurons each
- Architechture 5: 8 hidden layers with 100 neurons each

Case 1:

For this case, a stream-function forulation was used, were a function $\psi(x,t)$ was defined as

$$U = \nabla \times \psi(x,t)$$

therefore

$$u = \psi_y$$
$$v = -\psi_x$$

For this case, the following exact solution $\psi(x,t)$ and $p(x,t)$ was considered

$$\psi(x,t) = (1 - x^2y^2)e^{-t}$$
$$p(x,t) = xy$$

which, for $\nu = 1$, gives the PDE below

$$\begin{aligned} u_t + uu_x + vu_y - u_{xx} - u_{yy} + p_x &= 2x^2ye^{-t} + 12x^3y^2e^{-t} + 4ye^{-t} + y \\ v_t + uv_x + vv_y - v_{xx} - v_{yy} + p_y &= -2xy^2e^{-t} + 4x^2y^3e^{-t} - 4ye^{-t} + x \end{aligned} \tag{7}$$

When solving Equation 7 with the architectures explained above, the following results were obtained

	$u(x,t)$		$p(x,t)$	
time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0	1.7341×10^{-5}	0.8328	2.6630×10^{-2}	32.6373
0.1	1.1645×10^{-3}	6.8252	1.0959×10^{-2}	20.9369
0.2	3.8980×10^{-3}	12.4867	8.0858×10^{-3}	17.9842
0.3	7.4127×10^{-3}	17.2194	1.0201×10^{-2}	20.2000
0.4	1.1246×10^{-2}	21.2102	1.4190×10^{-2}	23.8244
0.5	1.5156×10^{-2}	24.6217	1.9096×10^{-2}	27.6374
0.6	1.9022×10^{-2}	27.5841	2.4764×10^{-2}	31.4732
0.7	2.2797×10^{-2}	30.1976	3.1217×10^{-2}	35.3368
0.8	2.6465×10^{-2}	32.5360	3.8418×10^{-2}	39.2008
0.9	3.0017×10^{-2}	34.6508	4.6221×10^{-2}	42.9982
1.0	3.3443×10^{-2}	36.5749	5.4403×10^{-2}	46.6490

Table 18: Results for the first architecture for the 2-dimesnional Navier Stokes Equations

	$u(x,t)$		$p(x,t)$	
time	**MSE**	**l^2 error**	**MSE**	**l^2 error**
0.0	9.9230×10^{-5}	1.9923	1.5095×10^{-2}	24.5721
0.1	8.7404×10^{-4}	5.9128	8.5973×10^{-3}	18.5443
0.2	2.6156×10^{-3}	10.2286	7.6538×10^{-3}	17.4972
0.3	5.0194×10^{-3}	14.1696	9.5980×10^{-3}	19.5939
0.4	7.8569×10^{-3}	17.7278	1.3048×10^{-2}	22.8455
0.5	1.0956×10^{-2}	20.9343	1.7481×10^{-2}	26.4430
0.6	1.4186×10^{-2}	23.8212	2.2916×10^{-2}	30.2760
0.7	1.7444×10^{-2}	26.4155	2.9685×10^{-2}	34.4588
0.8	2.0647×10^{-2}	28.7384	3.8272×10^{-2}	39.1266
0.9	2.3724×10^{-2}	30.8054	4.9202×10^{-2}	44.3630
1.0	2.6613×10^{-2}	32.6271	6.2971×10^{-2}	50.1880

Table 19: Results for the second architecture for the 2-dimesnional Navier Stokes Equations

	$u(x,t)$		$p(x,t)$	
time	**MSE**	l^2 **error**	**MSE**	l^2 **error**
0.0	2.1447×10^{-5}	0.9262	2.3688×10^{-2}	30.7818
0.1	1.1856×10^{-3}	6.8865	1.1292×10^{-2}	21.2527
0.2	3.8643×10^{-3}	12.4326	8.9173×10^{-3}	18.8863
0.3	7.1762×10^{-3}	16.9425	1.0742×10^{-2}	20.7285
0.4	1.0635×10^{-2}	20.6252	1.4303×10^{-2}	23.9189
0.5	1.4016×10^{-2}	23.6781	1.8837×10^{-2}	27.4495
0.6	1.7252×10^{-2}	26.2692	2.4307×10^{-2}	31.1812
0.7	2.0354×10^{-2}	28.5338	3.0877×10^{-2}	35.1438
0.8	2.3374×10^{-2}	30.5769	3.8664×10^{-2}	39.3266
0.9	2.6369×10^{-2}	32.4774	4.7642×10^{-2}	43.6540
1.0	2.9401×10^{-2}	34.2932	5.7633×10^{-2}	48.0137

Table 20: Results for the third architecture for the 2-dimesnional Navier Stokes Equations

	$u(x,t)$		$p(x,t)$	
time	**MSE**	l^2 **error**	**MSE**	l^2 **error**

Table 21: Results for the fourth architecture for the 2-dimesnional Navier Stokes Equations

	$u(x,t)$		$p(x,t)$	
time	**MSE**	l^2 **error**	**MSE**	l^2 **error**
0.0	1.6969×10^{-5}	0.8239	1.9618×10^{-2}	28.0128
0.1	9.4158×10^{-4}	6.1370	9.3919×10^{-3}	19.3824
0.2	3.1448×10^{-3}	11.2157	8.0040×10^{-3}	17.8930
0.3	5.9426×10^{-3}	15.4176	1.0275×10^{-2}	20.2732
0.4	8.9039×10^{-3}	18.8721	1.4243×10^{-2}	23.8686
0.5	1.1787×10^{-2}	21.7140	1.9205×10^{-2}	27.7166
0.6	1.4485×10^{-2}	24.0708	2.4883×10^{-2}	31.5488
0.7	1.6976×10^{-2}	26.0586	3.1112×10^{-2}	35.2773
0.8	1.9294×10^{-2}	27.7807	3.7755×10^{-2}	38.8615
0.9	2.1502×10^{-2}	29.3270	4.4696×10^{-2}	42.2827
1.0	2.3676×10^{-2}	30.7742	5.1843×10^{-2}	45.5382

Table 22: Results for the fifth architecture for the 2-dimesnional Navier Stokes Equations

5 Conclussions

Acknowledgements

References

[1] Logan JD. Applied partial differential equations. Springer; 2014.

[2] Farlow SJ. Partial differential equations for scientists and engineers. Courier Corporation; 1993.

[3] Strauss WA. Partial differential equations: An introduction. John Wiley & Sons; 2007.

[4] Hesthaven JS, Ubbiali S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. Journal of Computational Physics. 2018;363:55-78.

[5] Quarteroni A. Physics-Based and Data-Driven-Based Algorithms for the Simulation of the Heart Function; 2023. .

[6] Berg J, Nyström K. Neural networks as smooth priors for inverse problems for PDEs. Journal of Computational Mathematics and Data Science. 2021;1:100008.

[7] Meng X, Karniadakis GE. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. Journal of Computational Physics. 2020;401:109020.

[8] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics. 2019;378:686-707.

[9] Moschou SP, Hicks E, Parekh R, Mathew D, Majumdar S, Vlahakis N. Physics-Informed Neural Networks for modeling astrophysical shocks. Machine Learning: Science and Technology. 2023.

[10] Xiao MJ, Yu TC, Zhang YS, Yong H. Physics-informed neural networks for the Reynolds-Averaged Navier–Stokes modeling of Rayleigh–Taylor turbulent mixing. Computers & Fluids. 2023:106025.

[11] Lino M, Fotiadis S, Bharath AA, Cantwell CD. Current and emerging deep-learning methods for the simulation of fluid dynamics. Proceedings of the Royal Society A. 2023;479(2275):20230058.

[12] Zhu C, Byrd RH, Lu P, Nocedal J. Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization. ACM Trans Math Softw. 1997 dec;23(4):550–560. Available from: `https://doi.org/10.1145/279232.279236`.