PINNs for Parametric Incompressible Newtonian Flows

Rômulo M. Silva[1], Alvaro L. G. A. Coutinho[1]

[1]*Civil Engineering, COPPE/Federal University of Rio de Janeiro*
Rio de Janeiro, RJ 21941-598, Brazil
romulo.silva@nacad.ufrj.br, alvaro@nacad.ufrj.br

Abstract.

In this paper we demonstrate the application of Physics-Informed Neural Networks (PINNs) for learning the solution of the parametric steady incompressible Navier-Stokes equations for multiple flow regimes for the well-known channel-driven cavity flow problem, given only the geometry and boundary conditions.

Keywords: PINNs, Incompressible Flows, Parametric PDEs

1 Introduction

Physics-Informed Neural Networks (PINNs) have been successfully used for a wide range of problems, including, but not limited to, wave propagation, design optimization, fluid flows, Bayesian parameter estimation, and more. For recent reviews on the state-of-the-art of PINNs, see [1, 2]. Even though PINNs may not seem efficient for solving simple forward problems, PINNs shine when solving parametric and inverse PDE-driven problems [3, 4] that could be not only complicated to solve with traditional methods but impossible sometimes. Parametric PDEs arise (sometimes implicitly) in diverse areas in the Oil & Gas industry and are usually applied to the optimal systems design and simulation of multiple scenarios. The ultimate goal is to use a trained PINN as a surrogate of a PDE-driven problem, avoiding the necessity to compute costly solutions for multiple scenarios. This is particularly useful in field inspection and maintenance, where decisions must be taken quickly, possibly with the aid of mobile devices only.

The remainder of this work is organized as follows. The following section presents the steady incompressible parametric Navier-Stokes equations, giving particular attention to different forms of deriving the equations such that the continuity equation is automatically satisfied. Section 3 briefly reviews the basic PINNs theory and how they are applied to the equations described in Section 2. Numerical examples showing how the PINNs can learn the solution of the parametric steady incompressible Navier-Stokes equations for multiple flow regimes for the well-known channel-driven cavity flow problem are given in Section 4. The paper ends with a summary of our main conclusions.

2 Navier-Stokes Equations

The Navier-Stokes (NS) equations governing steady, incompressible fluid flows can be written in terms of different sets of variables. Changing the way we solve the Navier-Stokes equations may result in some benefits, such as the automatic satisfaction of the continuity equation. Further, the NS equations depend on a parameter, the Reynolds number, $Re = \frac{||\mathbf{u}||L}{\nu}$, where $||\mathbf{u}||$ is the velocity magnitude, L is a characteristic length, and ν is the kinematic viscosity of the fluid. Next, we briefly present the NS equations written in terms of their primitive variables, the well-known Stream function-Vorticity, and a different Stream function-Pressure formulation. Other formulations exist [5, 6], but we restrict ourselves to the ones below.

Primitive variables

The steady incompressible NS equations in primitive variable reads:

$$(\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u}+\boldsymbol{\nabla}p-\frac{1}{Re}\nabla^2\mathbf{u}=0,\quad \forall\mathbf{x}\in\Omega, \tag{1}$$

$$\boldsymbol{\nabla}\cdot\mathbf{u}=0,\quad \forall\mathbf{x}\in\Omega, \tag{2}$$

where $\mathbf{u}=\{u,v\}$ is the velocity vector, p is the pressure. Eqs. (1) and (2) are respectively the momentum and continuity equations. These equations are supplemented by proper Dirichlet, Neumann, or mixed (Robin) boundary conditions, that is, the known data.

The ψ – ω formulation (Stream function-Vorticity)

For a 2D flow, one may write the NS equations in terms of the vorticity $\boldsymbol{\omega}=\boldsymbol{\nabla}\times\mathbf{u}$ and the stream function ψ. By using this scheme, we can remove the necessity of including any information about the pressure p whose gradients are included in the momentum equations (Eq. (1)). Finally, this scheme exactly satisfies the continuity equation (Eq. (2)) due to the particular choice for the velocity field shown in Eq. (5). The $\psi-\omega$ equations are written as follows:

$$\mathbf{u}\cdot\boldsymbol{\nabla}\omega-\frac{1}{Re}\nabla^2\omega=0,\quad \forall\mathbf{x}\in\Omega, \tag{3}$$

$$\nabla^2\psi+\omega=0,\quad \forall\mathbf{x}\in\Omega, \tag{4}$$

$$\mathbf{u}=\left\{\frac{\partial\psi}{\partial y},-\frac{\partial\psi}{\partial x}\right\} \tag{5}$$

where ω is the $z-$component of the vorticity vector (assuming that the flow is in the $x-y$ plane), ψ is the stream function. If we need to recover the pressure field, one can solve the so-called Poisson Pressure Equation (PPE). Boundary conditions for the stream function and vorticity should be specified.

The ψ – p formulation (Stream function-Pressure)

Another alternative for solving a 2D flow problem with the continuity equation being exactly satisfied is to write the velocity components in terms of the stream function as shown in Eq. (7) and plug it into the momentum equations (Eq. (6)). We can then solve a system of two third-order PDEs in terms of ψ and p, where it is straightforward to derive the velocity components from the stream function. This scheme is then written as:

$$(\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{u}+\boldsymbol{\nabla}p-\frac{1}{Re}\nabla^2\mathbf{u}=0,\quad \forall\mathbf{x}\in\Omega, \tag{6}$$

$$\mathbf{u}=\left\{\frac{\partial\psi}{\partial y},-\frac{\partial\psi}{\partial x}\right\}. \tag{7}$$

The corresponding set of boundary conditions should be added to equations (6) and (7) to complete the problem description.

3 Physics-Informed Neural Networks

In this section we briefly introduce the Physics-Informed Neural Networks (PINNs) and how they can be used to learn the solution of the parametric steady incompressible NS equations for multiple flow regimes. Consider the problem presented in Eqs. (8)-(10) whose solution is given by $\mathbf{u}(\mathbf{x})$ with $\mathbf{x}\in\mathbb{R}^N$ and $\mathbf{u}\in\mathbb{R}^S$ with its physical parameters $\boldsymbol{\lambda}$ [3],

$$f\left(\mathbf{x};\mathbf{u};\frac{\partial\mathbf{u}}{\partial x_1},...,\frac{\partial\mathbf{u}}{\partial x_N};\frac{\partial\mathbf{u}^2}{\partial x_1x_1},...,\frac{\partial\mathbf{u}^2}{\partial x_1x_N};\boldsymbol{\lambda}\right)=0,\quad \forall\mathbf{x}\in\Omega,\Omega\subset\mathbb{R}^N, \tag{8}$$

$$\mathbf{u}(\mathbf{x})=g(\mathbf{x}),\quad \forall\mathbf{x}\in\Gamma_g,\Gamma_g\subset\mathbb{R}^{N-1}, \tag{9}$$

$$h\left(\mathbf{x};\frac{\partial\mathbf{u}}{\partial x_1},...,\frac{\partial\mathbf{u}}{\partial x_N};\boldsymbol{\lambda}\right)=0,\quad \forall\mathbf{x}\in\Gamma_h,\Gamma_h\subset\mathbb{R}^{N-1} \tag{10}$$

in which Eq. (8) is the governing PDE, Eq. (9) represents the *Dirichlet* boundary conditions, while Eq. (10) encapsulates both, *Neumann* and *Robin* boundary conditions. Furthermore, Ω denotes the internal domain, Γ_h and

Γ_g denote the natural and essential boundaries, respectively. Note that in the present case, the physical parameter is the Reynolds number, that is, $\boldsymbol{\lambda} = \{Re\}$.

In general, PDEs are difficult to solve and require approximation to be solved numerically. One can try to approximate the differential operator, but it is also possible to approximate the function space of the solution $\mathbf{u}$. The solution can be approximated using bases composed of monomial functions, piece-wise linear functions, or even by the space of the Neural Networks $\mathcal{H}_{NN}$. If we choose to approximate the solution $\mathbf{u}(\mathbf{x})$ by $\hat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta}) \in \mathcal{H}_{NN}$ in Eqs. (8)-(10) we obtain

$$f\left(\mathbf{x}; \frac{\partial \hat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta})}{\partial x_1}, ..., \frac{\partial \hat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta})}{\partial x_N}; \frac{\partial \hat{\mathbf{u}}^2(\mathbf{x};\boldsymbol{\theta})}{\partial x_1 x_1}, ..., \frac{\partial \hat{\mathbf{u}}^2(\mathbf{x};\boldsymbol{\theta})}{\partial x_1 x_N}; \boldsymbol{\lambda}\right) = \mathcal{R}(\mathbf{x};\boldsymbol{\theta};\boldsymbol{\lambda}) \tag{11}$$

where $\mathcal{R}(\mathbf{x};\boldsymbol{\theta};\boldsymbol{\lambda})$ is called the residual of the PDE and $\boldsymbol{\theta}$ are the parameters of the neural network.

For solving Eq. (8) using $\hat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta})$ as an approximation for its solution $\mathbf{u}(\mathbf{x})$, one should minimize the residual (Eq. (11)) at a set of *residual points* $\mathcal{T}_\mathcal{R} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x}_\mathcal{R}\}$ with $\mathcal{T}_\mathcal{R} \subset \Omega$ and also satisfy the boundary conditions with a acceptable accuracy at a set of points $\mathcal{T}_\mathcal{B} \subset \Gamma$. To measure how good is the approximation of the residual of the PDE we write,

$$\mathcal{L}_\mathcal{R}(\boldsymbol{\theta};\mathcal{T}_\mathcal{R}) = MSE\{\mathcal{R}(\mathbf{x};\boldsymbol{\theta};\boldsymbol{\lambda}), \mathbf{0}\}, \tag{12}$$

and for measuring the quality of the approximation of the boundary conditions

$$\mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta};\mathcal{T}_{\mathcal{BC}}) = MSE\left\{h\left(\mathbf{x}; \frac{\partial \mathbf{u}}{\partial x_1}, ..., \frac{\partial \mathbf{u}}{\partial x_N}; \boldsymbol{\lambda}\right), \mathbf{0}\right\} + MSE\{\hat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta}), g(\mathbf{x})\}, \tag{13}$$

where $MSE\{\mathbf{y}_{true}(\mathbf{x}), \mathbf{y}_{pred}(\mathbf{x})\}$ denotes the Mean Squared Error between the vectors $\mathbf{y}_{true}(\mathbf{x})$ and $\mathbf{y}_{pred}(\mathbf{x})$.

Notice that, Equation (13) involves not only the information about the boundary conditions, but also the initial conditions for time-dependent problems since the components of $\mathbf{x}$ denotes not only the spatial dimensions but also time. With these measures in hand, we can now write the final loss function for *informing* the physical laws to a neural network during its training phase,

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = w_\mathcal{R}\mathcal{L}_\mathcal{R}(\boldsymbol{\theta};\mathcal{T}_\mathcal{R}) + w_{\mathcal{BC}}\mathcal{L}_{\mathcal{BC}}(\boldsymbol{\theta};\mathcal{T}_{\mathcal{BC}}) \tag{14}$$

where $w_\mathcal{R}$ and $w_{\mathcal{BC}}$ are the weights of the loss function, which play an essential role in the minimization of the loss function.

4 Results

The problem consists of solving a 2D incompressible flow problem governed by the NS equations, and it is inspired by the one solved by Mahmood et al.[7] with a parabolic $x-$velocity profile at the inlet with $u_{max} = 0.3$, no-slip boundary conditions for the top and bottom walls, and a do-nothing boundary condition for the outlet. In [7] this problem is solved for $Re = 10, 20, 50$, and 100, while we solve for all the possible Reynolds numbers such that $Re \in [10, 100]$. We use a PINN that is expected to predict the solution of the NS equation at any spatial coordinates, given the Reynolds number for a situation in which the only data available consists of the geometry and boundary conditions. The geometry of the problem is shown in Fig. 1, as well as the residual and boundary points distribution. For comparison, we first solve the proposed problem for $Re = 10, 20, 50$, and 100 using the Finite Element Method (FEM) with Taylor-Hood elements. The FEM solver is implemented using Firedrake [8–12]. The FEM solution is shown in Fig. 2, and it will be assumed to be the *ground truth*. Notice that the ground truth will **not** be used as training data for the PINNs.

In contrast to the FEM solution which is given for a finite set of mesh points and needs to be completely re-solved for different Reynolds numbers, we show a PINN-based alternative for solving the parametric NS equations. The PINN will learn the problem solution at any coordinates x and y, given the Reynolds number Re. If one wants to solve the parametric NS equations through the ψ – ω scheme (Eqs. (3) - (5)), the solution for the parametric NS equations would then be given as $\psi = \psi(x, y; Re)$ and $\omega = \omega(x, y; Re)$ as shown in Fig. 3a. Similarly, the solution for the parametric NS equations using the ψ – p formulation would be given as $\psi = \psi(x, y; Re)$ and $p = p(x, y; Re)$ as shown in Fig. 3b. Figure 3 illustrates for both formulations how the PINN scheme works, showing the interaction between the PINN approximations and the PDEs for each scheme.

For both schemes in Figure 3, we use a neural network with 4 hidden layers and 32 neurons in each layer. We use the hyperbolic tangent as the activation function and a Fourier features map [13] in the form,

$$\gamma(x, y; Re) = [x, y, Re, \cos 2\pi x, \cos 2\pi y, \sin 2\pi x, \sin 2\pi y]^T, \tag{15}$$

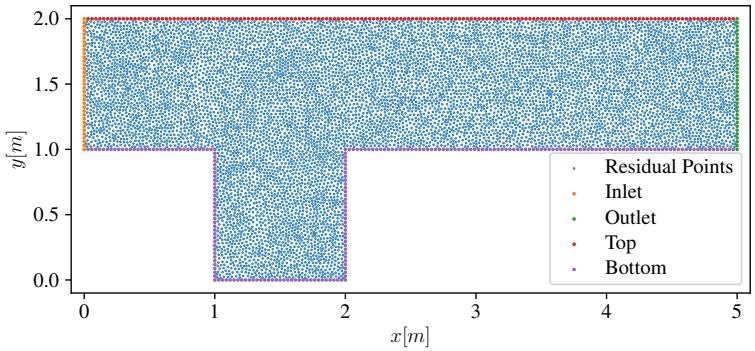


Figure 1. Channel-driven cavity flow problem: Boundary and residual evaluation points.

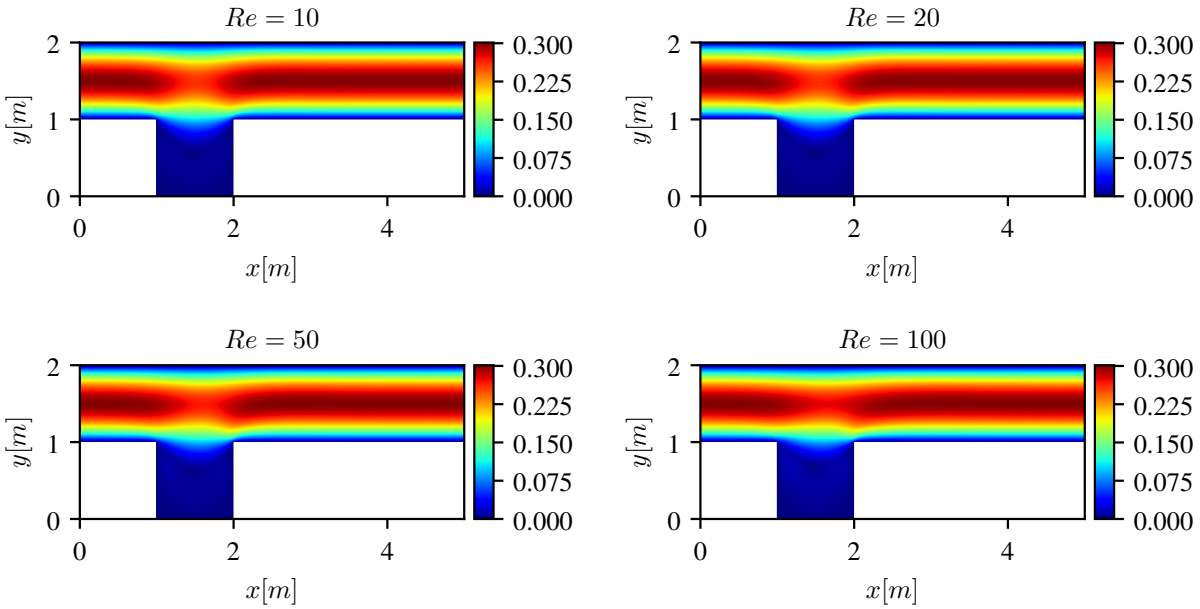
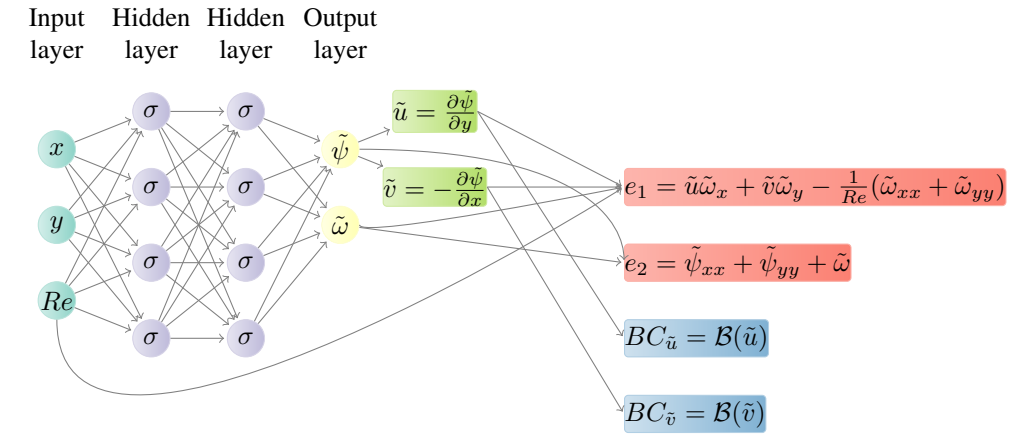


Figure 2. Channel-driven cavity flow problem: Ground truth solutions for $Re = 10, 20, 50$, and 100.

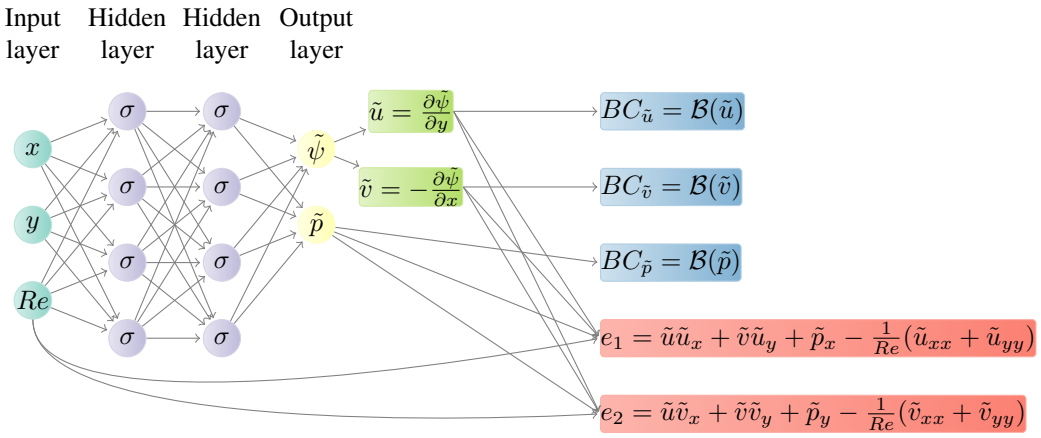
which contributes to improve the convergence in our experiments. In each iteration we use full batch for the residual (e_1, e_2) and boundary ($\mathcal{BC}_{\tilde{u}}$ and $\mathcal{BC}_{\tilde{v}}$) points x and y components in Figure 3, while the Re component is always sampled as $\mathcal{U}(10, 100)$. Therefore, the PINN is trained to solve the flow problem for **ALL** $Re \in [10, 100]$ at the same time. As the optimizer, we use the RMSProp optimizer with a fixed learning rate of 10^{-3} and let the model train for $800,000$ iterations. Our code is implemented using TensorFlow $2.4.1$ [14] and we enable the XLA compiler, which speeds-up the training process 4 to 5 times.

Along the training process, we compare the predictions of our model against the ground truth solutions for $Re = 10, 20, 50$, and 100, so we can see how fast the PINN learns to solve the parametric NS equations as shown in Fig. 4, that depicts the R^2 score history for both NS formulations. Notice that, for this particular problem, both schemes have more difficulty in solving the $y-$velocity component. Furthermore, as expected, the solution becomes more difficult as we increase the Reynolds number. At the end of the training process, we also evaluate the R^2 score for all the configurations, and it turns out that the worst result is obtained from the approximation of the $y-$velocity component for $Re = 100$ whose R^2 score is equal to 0.93. It is also worth noticing that the $\psi - p$ PINN scheme converges slower than the $\psi - \omega$ scheme, particularly for the $y-$velocity component.

In Figure 5 we can see the magnitude of the velocity component error in the domain at $Re = 100$. The error is the absolute value of the difference between the ground truth and the PINN solution. Errors are small but with peaks at some regions in the domain. The regions with high errors are larger for the $\psi - p$ formulation when compared with those obtained with the $\psi - \omega$ formulation. For the $\psi - \omega$ formulation, the high errors are concentrated in the right cavity corner.



(a) Scheme for the ψ – ω form.



(b) Scheme for the ψ – p form.

Figure 3. PINN schemes for solving the Parametric Navier-Stokes Equations.

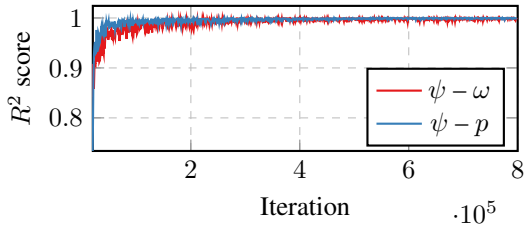
5 Conclusions

We find that even though PINNs take considerably longer to train when it comes to solving the parametric NS equations, the advantages of instantaneously predicting the solution given a triplet $\{x, y, Re\}$ surpass the burden of the training process. If we take, for instance, our worst result, predicting the $y-$velocity for $Re = 100$, its R^2 score achieves the mark of 0.9 within almost $380,000$ iterations for the ψ – ω formulation, which in our experiments would take the same amount of time for solving the problem 30 times for $Re = 100$ using Firedrake. Furthermore, once the PINN is trained, the time for making the predictions for any $\{x, y, Re\}$ is negligible. We can do thousands of predictions using the PINN while we solve a single problem using traditional methods.

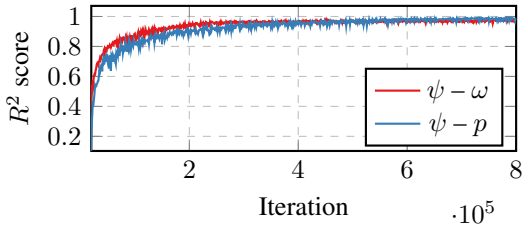
In addition, because of the 3rd order derivatives of ψ w.r.t. x and y in the ψ – p formulation, it takes ≈ 4 times more to perform an iteration when compared with the ψ – ω formulation, not to mention the lower quality of the solution.

As future works, we envisage trained PINNs immersed in visualization software as quick field inspection and maintenance tools, possibly in mobile devices such as tablets and phones. The extension of such use to more complex situations involving unsteady, non-Newtonian flows with parametrized geometries is also worth mentioning.

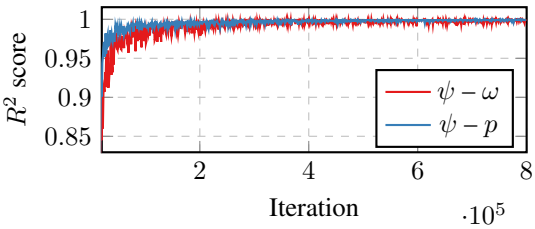
Acknowledgements. This work is funded by CNPq, FAPERJ (Center of Excellence in Digital Transformation and Artificial Intelligence of Rio de Janeiro State: Thematic Network in Renewable Energy and Climate Change). R. M. Silva is supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) -



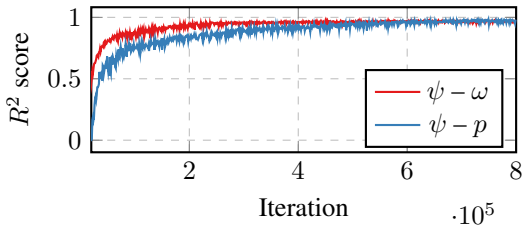
(a) R^2 score evolution for the $x-$velocity and $Re = 10$.



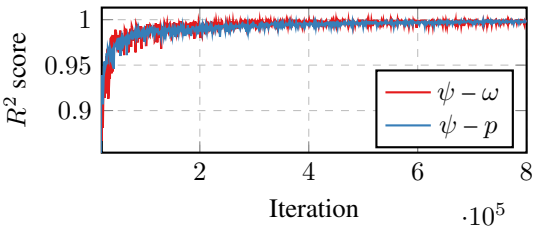
(b) R^2 score evolution for the $y-$velocity and $Re = 10$.



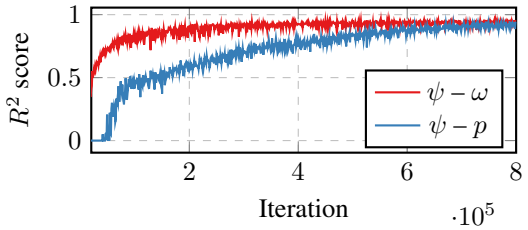
(c) R^2 score evolution for the $x-$velocity and $Re = 20$.



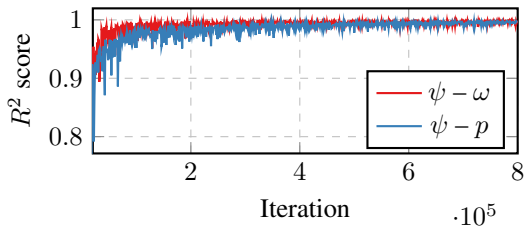
(d) R^2 score evolution for the $y-$velocity and $Re = 20$.



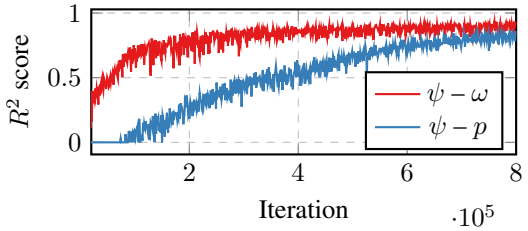
(e) R^2 score evolution for the $x-$velocity and $Re = 50$.



(f) R^2 score evolution for the $y-$velocity and $Re = 50$.



(g) R^2 score evolution for the $x-$velocity and $Re = 100$.



(h) R^2 score evolution for the $y-$velocity and $Re = 100$.

Figure 4. R^2 score evolution for several Reynolds numbers.

Finance Code 001.

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, vol. 3, n. 6, pp. 422–440, 2021.

[2] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *arXiv:2105.09506*, 2021.

[3] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, vol. 63, n. 1, pp. 208–228, 2021.

[4] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, M. Rietmann, del J. Aguila Ferrandis, W. Byeon, Z. Fang, and S. Choudhry. Nvidia simnet^{TM}: an ai-accelerated multi-physics simulation framework. *arXiv:2012.07938*, 2020.

[5] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. NSFnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, vol. 426, pp. 109951, 2021.

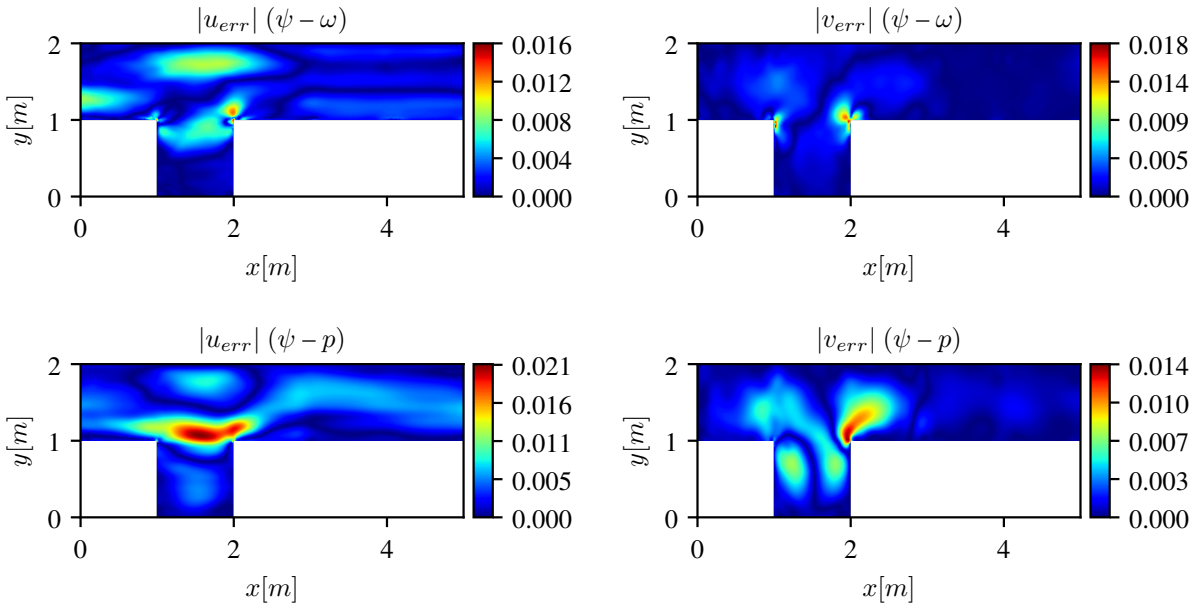


Figure 5. Magnitude of velocity component pointwise error for $Re = 100$. Top:$\psi - \omega$, bottom $\psi - p$

[6] D. Young, C. Tsai, and C. Wu. A novel vector potential formulation of 3d navier–stokes equations with through-flow boundaries by a local meshless method. *Journal of Computational Physics*, vol. 300, pp. 219–240, 2015.

[7] R. Mahmood, S. Bilal, A. H. Majeed, I. Khan, and E.-S. M. Sherif. A comparative analysis of flow features of Newtonian and power law material: A New configuration. *Journal of Materials Research and Technology*, vol. 9, n. 2, pp. 1978–1987, 2020.

[8] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly. Firedrake: automating the finite element method by composing abstractions. *ACM Trans. Math. Softw.*, vol. 43, n. 3, pp. 24:1–24:27, 2016.

[9] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, vol. 34, n. 9, pp. 1124–1139. New Computational Methods and Software Tools, 2011.

[10] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, eds, *Modern Software Tools in Scientific Computing*, pp. 163–202. Birkhäuser Press, 1997.

[11] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.11, Argonne National Laboratory, 2019.

[12] R. C. Kirby and L. Mitchell. Solver composition across the PDE/linear algebra barrier. *SIAM Journal on Scientific Computing*, vol. 40, n. 1, pp. C76–C98, 2018.

[13] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv:2006.10739*, 2020.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org, 2015.