Optimizing a Physics-Informed Neural Network to solve the Reynolds Equation

Z. Sánchez López and G. Berenice Díaz Cortés
Instituto Mexicano del Petróleo,
Eje Central Lázaro Cárdenas Norte 152, Col. San Bartolo Atepehuacán, Gustavo A. Madero, 07730, Ciudad de México.

Received 16 February 2024; accepted 8 August 2024

This study focuses on the optimization of a Physics-Informed Neural Network (PINN) to address Partial Differential Equation (PDE) problems associated with fluid flow. Specifically, the stationary, one-dimensional classical Reynolds equation is solved using the PINN. Within the conducted studies, a comparison is made between the solutions obtained using the PINN, the numerical Finite Difference Method (FD), and the analytical solution. We study various scenarios with diverse hyper-parameters such as learning rate, epochs, number of training points, etc., for constructing the neural network to identify the optimal setup. The PINN accurately approximated the solution to the Reynolds equation (up to $\mathcal{O}(10^{-2})$. This suggests that PINNs can be used to address diverse problems in fluid dynamics. We proposed a PINN configuration that outperformed the PINN presented in the literature. The finite differences method obtains a better approximation than the PINNs, however, the full potential of the PINNs is yet to be determined, as it can include data from the problem, that finite difference method (FD) can not. Further studies are planned to investigate the capabilities of PINNs.

Keywords: PINN; PDE solution; Reynolds equation; fluid dynamics; hyperparameters optimization.

DOI: https://doi.org/10.31349/RevMexFis.71.020601

1. Introduction

Physics-informed neural networks (PINNs) are a state-of-the-art framework recently developed to solve partial differential equations [1] for both, forward and inverse problems.

PINNs were introduced in 2017 as a new class of data-driven solvers in a two-part paper [2,3] published in a merged version later in 2019, Raissi *et al*. [4] developed the PINNs that can accurately solve both forward problems that approximate the solutions of the governing mathematical model, as well as inverse problems, where the model parameters are inferred from the observed data. The PINN algorithm inputs the governing equation into the network and thus enriches the loss function by adding a residual form of the equation, which essentially acts as a penalizing term to constrain the space of admissible solutions. In this setting, the problem of inferring solutions of PDEs is transformed into an optimization problem of the loss function that contains the physics of the problem in the residual, boundary, and initial conditions loss terms [5].

PINNs have now been applied to solve a variety of problems including fluid mechanics [4,6-12], solid mechanics [13-16], heat transfer [17-19], geophysics [9,20-22], and flow in porous media [23-29].

Applications in flow simulations include solving Navier-Stokes equations [30], fluid flow through random heterogeneous media [31-33], Reynolds-Averaged Navier-Stokes simulations [34-36], among others [37].

The performance of the PINNs is influenced by various factors, including data quantity, quality, and features. A key factor is the configuration of parameters based on the problem specification. Parameters in a PINN can be classified into two types: model parameters, which are estimated from data such as weights and biases, and hyperparameters, which are external and cannot be estimated directly from data, such as the learning rate, number of neurons and layers, etc.

Optimizing hyperparameters is essential for maximizing model performance, as the best values are those that yield the highest accuracy. Various methods, such as grid search, randomized search, genetic and evolutionary algorithms, and Bayesian optimization, can be employed to identify the best hyperparameter values [38-41].

In this paper, we optimize a Physics-Informed Neural Network (PINN) to solve the stationary one-dimensional Reynolds equation, as previously presented in the literature [38]. We complement this work by providing a detailed explanation of the analytically computed derivatives required for training the PINN, aiming for a clearer understanding of the process (see Appendix A).

As we will see in the first experiment, the PINN does not always converge to the solution, therefore, an statistical study is performed. Due to the instability of the particular PINN we are working on, automatized optimization methods that include large number of computations such as grid search, evolutionary algorithms and Bayesian optimization are not the best option due to the number of repetitions performed for the statistics, and are out of the scope of this paper. Instead, we selected a set of parameters to perform the study based on the best observed performance, more details are presented in the Sec. 3.

This paper is distributed as follows, in Sec. 2, we present the methodology for applying PINNs to solve PDEs, followed by its specific application in solving the Reynolds Equation, discussed in Subsec. 2.1. We present the neural network architecture used to solve the Reynolds equation in Subsec. 2.2. Then, in Subsec. 2.3, we provide a brief description of the experiments conducted, where the network's hyperparameters were adjusted for optimization. Finally, in

Sec. 3, we present the results and discuss their implications, finally, we include the conclusions of this work.

The experiments presented in this work were performed in python in an Intel@ Core i7-7700HQ CPU @2.80 GHz x8, with 32 GB of RAM. The PINN for the Reynolds Equation theory and code is available at `https://github.com/gbdiazc/Reynolds_PINN_Notebook.git`.

2. Methodology of physics-informed neural networks for solving PDEs

The methodology to approximate a PDE with physics-informed neural networks (PINNs) is based on the minimization of the cost function, taking into account the residual form of the PDE, together with the boundary conditions, and initial conditions [4]. The residual form of the PDE is given in Eq. (1),

$$R(t,\boldsymbol{x}) = \frac{\partial \boldsymbol{u}(\boldsymbol{x},t)}{\partial t} + \mathcal{N}[\boldsymbol{u}(\boldsymbol{x},t)] = 0, \qquad \boldsymbol{x} \in \boldsymbol{\Omega},\ t \in [0,T], \tag{1}$$

subject to the initial Eq. (2) and boundary conditions Eq. (3):

$$\boldsymbol{u}(\boldsymbol{x}, t=0) = g(\boldsymbol{x}), \qquad \boldsymbol{x} \in \boldsymbol{\Omega} \tag{2}$$

$$\mathcal{B}[\boldsymbol{u}(\boldsymbol{x},t)] = 0, \qquad \boldsymbol{x} \in \boldsymbol{\partial}\Omega,\ t \in [0,T]. \tag{3}$$

In Eq. (1), $\mathcal{N}[\cdot]$ represents a differential operator (linear or nonlinear), and $\mathcal{B}[\cdot]$ is an operator representing the boundary conditions (Dirichlet, Neumann, Robin, or periodic). The vector $\boldsymbol{x} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_d}) \in \boldsymbol{\Omega}$ is the spatial coordinate, that in general could be in $\mathbb{R}^d$, and $t \in [0,T]$ is the temporal coordinate. The spatial domain and the boundary of the domain where the PDE is defined are represented by $\boldsymbol{\Omega}, \boldsymbol{\partial\Omega} \in \mathbb{R}^d$, respectively.

A feed-forward neural network has an architecture such that the information flows in a single direction. It consists of the input layer, hidden layers, and an output layer (See Fig. 1). The input layer corresponds to the data inputted to the NN. If the PDE depends on the variables $\boldsymbol{x}, t$, a set of points $\boldsymbol{x_i}, t_i$ in the domain $\boldsymbol{\Omega}$ are selected to train the network, the output of this layer is the original data $z^0 = (\boldsymbol{x}, t)$.

Next, we have the hidden layers, where all the computation is done. Neurons in adjacent layers are connected but neurons within a single layer share no connection, and each neuron in layer (L) is connected to all neurons in the next layer $(L+1)$. In each of these layers, the input data (z^{L-1}) of the previous layer $(L-1)$ is multiplied by the weights and the bias is added, therefore the output of each layer is given by $\boldsymbol{z}^L = \boldsymbol{w}^L \boldsymbol{z}^{L-1} + \boldsymbol{b}^L$. The output of each layer passes through an activation function φ, before moving to the next layer *i.e.*, $\boldsymbol{z}^L = \varphi(\boldsymbol{w}^L \boldsymbol{z}^{L-1} + \boldsymbol{b}^L)$.

In the last layer, the expected approximation is obtained. For a neural network of Nn layers the output of the previous layer $Nn-1$ is multiplied by the weights and biases and an activation function is applied, leading to the expected solution $u_\theta(\boldsymbol{x},t) = \varphi(\boldsymbol{w}^{Nn} \boldsymbol{z}^{Nn-1} + \boldsymbol{b}^{Nn})$. The layers can be summarized as:

$$\begin{aligned}
&\text{Input:} \quad (Data) \quad \boldsymbol{z}^0 = (\boldsymbol{x}, t). \\
&\text{Hidden:} \quad \boldsymbol{z}^L = \varphi(\boldsymbol{w}^L \boldsymbol{z}^{L-1} + \mathbf{b}^L), \quad 1 \le L \le Nn-1. \\
&\text{Output:} \quad u_\theta(\boldsymbol{x},t) = \boldsymbol{z}^{Nn} = \varphi(\boldsymbol{w}^{Nn} \boldsymbol{z}^{Nn-1} + \boldsymbol{b}^{Nn}).
\end{aligned} \tag{4}$$

The solution $\boldsymbol{u}(t,\boldsymbol{x})$ of the PDE is approximated with the output of a Neural Network $\boldsymbol{u}_\theta(t,\boldsymbol{x})$, where $\boldsymbol{\varphi(.)}$ is an activation function that can be linear or nonlinear, $\mathbf{w}^L$ and $\mathbf{b}^L$ are the weights and biases of the L^{th} layer, and together, they form the parameter vector of the neural network approximation denoted by $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \{\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^L, \boldsymbol{b}^1, \boldsymbol{b}^2, \ldots, \boldsymbol{b}^L\}. \tag{5}$$

The parameters of the neural network are randomly initialized and updated iteratively by minimizing a loss function.

The architecture of the PINNs differs from the architecture of a traditional NN in the last part of the network. The PINN works in the first part as a NN, however, the PINN takes the output $\boldsymbol{u}_\theta$ and computes its derivatives using the given equations (see Fig. 2). It calculates the loss functions related to the PDE, initial and boundary conditions, and, if available, due to the data. The final step involves the feed-

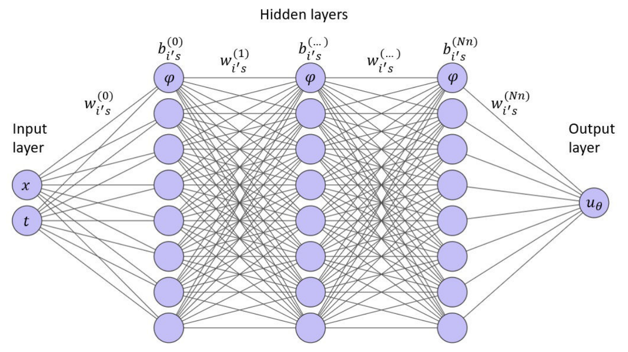

FIGURE 1. Architecture of a Feed Forward Neural Network, image created in Ref. [39] and adapted.

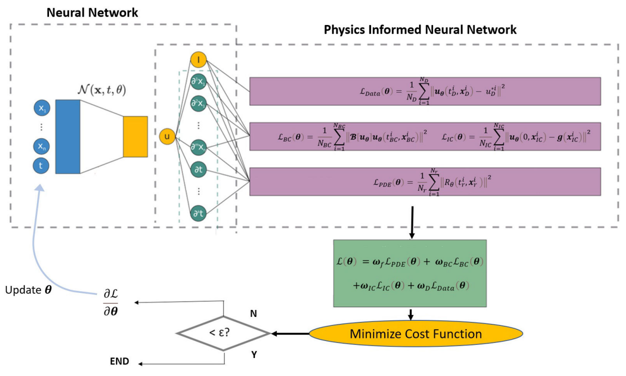
Neural Network
Physics Informed Neural Network
$\mathcal{N}(\mathbf{x},t,\theta)$
x_1
x_n
t
u
I
$\partial^1 x_1$
$\partial^2 x_1$
$\partial^m x_n$
∂t
$\partial^l t$
$\mathcal{L}_{Data}(\boldsymbol{\theta}) = \frac{1}{N_D}\sum_{i=1}^{N_D} \|\boldsymbol{u_\theta}(t_D^i, \boldsymbol{x}_D^i) - u_D^{*i}\|^2$
$\mathcal{L}_{BC}(\boldsymbol{\theta}) = \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}} \|\mathcal{B}[\boldsymbol{u_\theta}]\boldsymbol{u_\theta}(t_{BC}^i, \boldsymbol{x}_{BC}^i)\|^2$
$\mathcal{L}_{IC}(\boldsymbol{\theta}) = \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}} \|\boldsymbol{u_\theta}(0, \boldsymbol{x}_{IC}^i) - \boldsymbol{g}(\boldsymbol{x}_{IC}^i)\|^2$
$\mathcal{L}_{PDE}(\boldsymbol{\theta}) = \frac{1}{N_r}\sum_{i=1}^{N_r} \|R_\theta(t_r^i, \boldsymbol{x}_r^i)\|^2$
$\mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{\omega}_f \mathcal{L}_{PDE}(\boldsymbol{\theta}) + \boldsymbol{\omega}_{BC}\mathcal{L}_{BC}(\boldsymbol{\theta}) + \boldsymbol{\omega}_{IC}\mathcal{L}_{IC}(\boldsymbol{\theta}) + \boldsymbol{\omega}_D \mathcal{L}_{Data}(\boldsymbol{\theta})$
Minimize Cost Function
$< \varepsilon?$
N
Y
END
Update $\boldsymbol{\theta}$
$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$

FIGURE 2. Building Blocks of Physics-Informed Neural Networks (PINNs), modified Image of [40].

back mechanism, which minimizes the loss using an optimizer with a specified learning rate to obtain the neural network parameters $\boldsymbol{\theta}$ [40].

The loss function takes into account four error terms, see Eq. (7). The first term is related to the PDE ($\mathcal{L}_{PDE}$), to compute this term, we input the approximate solution $\boldsymbol{u}_\theta(t,\boldsymbol{x})$ into the residual Eq. (1) as [41]:

$$R_\theta(t,\boldsymbol{x}) = \frac{\partial \boldsymbol{u}_\theta}{\partial t}(t_r,\boldsymbol{x}) + \mathcal{N}\left[\boldsymbol{u}_\theta\right](t_r,\boldsymbol{x}). \tag{6}$$

The second term, takes into account the initial conditions ($\mathcal{L}_{IC}$), if the problem requires them. The third term is related to the boundary conditions ($\mathcal{L}_{BC}$), and the last term is related to the observation data ($\mathcal{L}_{Data}$) if they are available.

Then, a physics-informed model can be trained by minimizing the following composite loss function:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \boldsymbol{\omega}_f\mathcal{L}_{PDE}(\boldsymbol{\theta}) + \boldsymbol{\omega}_{BC}\mathcal{L}_{BC}(\boldsymbol{\theta})\\ &+ \boldsymbol{\omega}_{IC}\mathcal{L}_{IC}(\boldsymbol{\theta}) + \boldsymbol{\omega}_{D}\mathcal{L}_{Data}(\boldsymbol{\theta}),\end{aligned} \tag{7}$$

where $\boldsymbol{\omega}_f, \boldsymbol{\omega}_{BC}, \boldsymbol{\omega}_{IC}$ and $\boldsymbol{\omega}_D$ are weights associated with the loss terms. In this work, $\boldsymbol{\omega}_f = \boldsymbol{\omega}_{BC} = \boldsymbol{\omega}_{IC} = \boldsymbol{\omega}_D = 1$.

In this work, the mean square error (MSE) is employed to compute the losses in Eq. (8). These terms are calculated as follows [40-42]:

$$\begin{aligned}\mathcal{L}_{PDE}(\boldsymbol{\theta}) &= \frac{1}{N_r}\sum_{i=1}^{N_r}\left\|R_\theta(t_r^i,\boldsymbol{x}_r^i)\right\|^2,\\ \mathcal{L}_{IC}(\boldsymbol{\theta}) &= \frac{1}{N_{IC}}\sum_{i=1}^{N_{IC}}\left\|\boldsymbol{u}_{\boldsymbol{\theta}}(0,\boldsymbol{x}_{IC}^i) - \boldsymbol{g}(\boldsymbol{x}_{IC}^i)\right\|^2,\\ \mathcal{L}_{BC}(\boldsymbol{\theta}) &= \frac{1}{N_{BC}}\sum_{i=1}^{N_{BC}}\left\|\mathcal{B}[\boldsymbol{u}_{\boldsymbol{\theta}}](t_{BC}^i,\boldsymbol{x}_{BC}^i)\right\|^2,\\ \mathcal{L}_{Data}(\boldsymbol{\theta}) &= \frac{1}{N_D}\sum_{i=1}^{N_D}\left\|\boldsymbol{u}_\theta(t_D^i,\boldsymbol{x}_D^i) - u_D^{*i}\right\|^2,\end{aligned} \tag{8}$$

where $\{t_r^i,\boldsymbol{x}_r^i\}_{i=1}^{N_r}$ are the selected points, sampled randomly in $\boldsymbol{\Omega}$, also known as collocation points used to evaluate the PDE residual equation R_θ, $\{\boldsymbol{x}_{IC}^i\}_{i=1}^{N_{IC}}$ are the points where the initial conditions are imposed, $\{t_{BC}^i,\boldsymbol{x}_{BC}^i\}_{i=1}^{N_{BC}}$ are thepoints in the boundary conditions, and $\mathcal{L}_{Data}(\boldsymbol{\theta})$ isthe difference between the approximation $\boldsymbol{u}_\theta$ and the labels or solutions to the equation u_D^{*i} of the training data for N_D labeled data points.

To update the parameters $\boldsymbol{\theta}$ the gradient descent is used in this work, given by:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_n), \tag{9}$$

where η is the learning rate, a hyperparameter that depends on each network, and $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_n)$ are the derivatives of the loss function. More details are presented below for the studied case.

2.1. PINNs applied to solving the Reynolds Equation

In this work, the optimization of a Physics-Informed Neural Network (PINN) will be conducted to solve the classical one-dimensional steady-state Reynolds equation in the interval $\Omega = [0,1]$.

The dimensionless form of the Reynolds equation for the steady-state, one-dimensional case is expressed as follows Eq. (10) [38,43]:

$$\frac{d}{dx}\left(c(x)\frac{dy}{dx}\right) = c^{'}(x)y^{'} + c(x)y^{''} = f(x);\quad 0 < x < 1, \tag{10}$$

where

$$\begin{aligned}c(x) &= H^3;\quad H(x) = 1 + K - Kx,\\ c^{'}(x) &= -3K(1+K-Kx)^2,\\ f(x) &= \frac{dH}{dx} = -K,\end{aligned} \tag{11}$$

with the following boundary conditions:

$$y(0) = 0,\quad y(1) = 0. \tag{12}$$

Equation (10) has an analytical solution given by:

$$y_{ex}(x) = \frac{1}{K}\left(\frac{1}{1+K-Kx} - \frac{1+K}{2+K}\frac{1}{(1+K-Kx)^2} - \frac{1}{2+K}\right). \tag{13}$$

If we define the operators M and N as:

$$My = c'(x)y' + c(x)y'', \tag{14}$$

$$Ny = \begin{bmatrix}y(0)\\ y(1)\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix}. \tag{15}$$

Equations (10) and (12) can be rewritten in terms of the operators M and N for a given point x in the domain $x \in [0,1]$, as follows:

$$My - f = 0,\qquad 0 < x < 1, \tag{16}$$

$$Ny - \mathbf{b} = 0, \tag{17}$$

where **b = 0**.

In the PINNs approach, we approximate the solution with the neural network, that is:

$$y(x) \approx y_\theta(x;\boldsymbol{\theta}). \tag{18}$$

In this particular case, the cost function consists of the sum of two error terms, the term due to the residual equation of the PDE, and the term due to the boundary conditions, and it is expressed as follows:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{PDE} + \mathcal{L}_{BC} = \left\langle (My - f)^2 \right\rangle + \left((Ny - \mathbf{b}) \cdot \mathbf{e}_1\right)^2 + \left((Ny - \mathbf{b}) \cdot \mathbf{e}_2\right)^2, \tag{19}$$

where

$$\mathcal{L}_{PDE} = \left\langle (My - f)^2 \right\rangle = \left\langle \left(c'(x)y' + c(x)y'' - f\right)^2 \right\rangle, \tag{20}$$

and

$$\mathcal{L}_{BC} = (y(0))^2 + (y(1))^2. \tag{21}$$

Substituting (20) and (21) into (19), the cost function takes the form:

$$\mathcal{L}(\boldsymbol{\theta}) = \left\langle (c'(x)y' + c(x)y'' - f)^2 \right\rangle + (y(0))^2 + (y(1))^2. \tag{22}$$

This cost function is minimized with the gradient descent algorithm to update the weights and biases iteratively as indicated in Eq. (9).

2.2. Architecture of the PINN

The architecture of the PINN developed here has an input node x (the independent variable representing spatial coordinate), a hidden layer consisting of Nn nodes, and an output node $y(x)$ (the dependent variable representing pressure).

Figure 3 shows a graphical illustration of the PINN architecture used to solve the Reynolds problem.

The prediction of the PINN, $y(x)$, is obtained as follows: the input x is multiplied by the initial weights $\boldsymbol{w}_i^{(0)}$ and the initial biases, $\boldsymbol{b}_i^{(0)}$, are added to this result. Later, the activation function is applied and the outcome is then multiplied by the second set of weights $\boldsymbol{w}_i^{(1)}$ and a second bias $b^{(1)}$ is added to it:

$$y(x) = \sum_{i=1}^{N_n} w_i^{(1)} \varphi(w_i^{(0)} x + b_i^{(0)}) + b^{(1)}. \tag{23}$$

We use as activation function the sigmoid function, defined as:

$$\varphi(\varepsilon) = \frac{1}{1 + e^{-\varepsilon}}, \quad \mathbb{R} \to [0, 1]. \tag{24}$$

To update the weights and biases iteratively, we use the gradient descent algorithm, which updates the weights and biases as follows:

$$w_{i+1}^{(0)} = w_i^{(0)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial w_i^{(0)}}, \quad w_{i+1}^{(1)} = w_i^{(1)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial w_i^{(1)}},$$

$$b_{i+1}^{(0)} = b_i^{(0)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial b_i^{(0)}}, \quad b_{i+1}^{(1)} = b_i^{(1)} - \eta \frac{\partial \mathcal{L}(\theta)}{\partial b_i^{(1)}}, \tag{25}$$

where η is a hyperparameter known as the learning rate.

To compute the gradients, the cost function, Eq. (22), the output Eq. (23) of the PINN is evaluated at a set of points in the domain to compute the residual equation, and the boundary conditions $y(0)$, $y(1)$, its first derivative $y^{'}(x)$ and second derivative $y^{''}(x)$ in terms of the weights and biases. The derivatives are computed by deriving the equations analytically, see Appendix A for details.

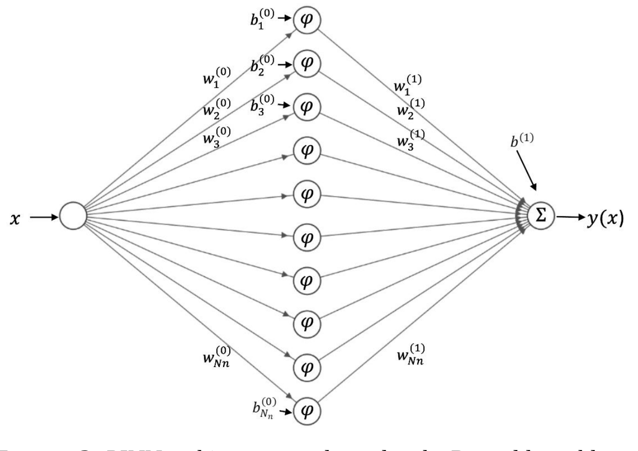

FIGURE 3. PINN architecture used to solve the Reynolds problem, image created in Ref. [39] and adapted.

2.3. Numerical experiments

To optimize the PINN, we performed five experiments designated as Experiment A, B, C, D, and E, with the first experiment being Experiment A, and so forth. In Experiment A, we replicate the results obtained by Andreas Almqvist (2021) [38]. In the subsequent experiments, we optimize the PINN by adjusting the following hyperparameters, Experiment B: the learning rate (lr) and batch size (Tb), Experiment C: the number of points in the domain (Ni), Experiment D: the number of epochs (Ne), and Experiment E: the number of neurons (Nn) in Experiment E.

Below, we provide a detailed description of each of the aforementioned experiments, and we present the results and subsequent discussion.

3. Results and discussion

In this section, we describe and analyze the five experiments (Experiment A, Experiment B, Experiment C, Experiment D, and Experiment E) conducted to optimize the PINN. In all cases, a fully connected feedforward neural network was used, with a sigmoid activation function and the loss function defined by Eq. (22). The architecture of the PINN consists of an input layer with one neuron, a hidden layer with 10 neurons (except in Experiment E, where the number of neurons in the hidden layer was modified), and an output layer with one neuron. The PINN was trained using the Gradient Descent optimizer.

3.1. Experiment A

In this section, we repeat the experiment performed by Andreas Almqvist (2021) [38], and compare it with the analytical solution, Eq. (13). The PINN was trained with the same hyperparameters as in Ref. [38] presented in Table I, and the architecture is described in Secs. 2.1 and 2.2. The PINN with this set of hyperparameters and architecture is now on referred to as PINN-2.

The comparison between the predictions of PINN-2 and the analytical solution (13) within the domain [0,1] is depicted in Fig. 4 for two scenarios, identified as "Prediction 1" (represented by orange points) and "Prediction 2" (represented by green points), respectively. They are contrasted with the analytical solution (illustrated by a blue solid curve).

We notice that PINN Prediction 1 fits with the analytical solution, while PINN Prediction 2 does not fit. In Fig. 5, the convergence of the cost function is presented for each prediction, both in the case of PINN Prediction 1 (orange) and the case of PINN Prediction 2 (green). We can observe that for PINN Prediction 1, its cost function converges to a value closer to zero than in the case of PINN Prediction 2. Note that no changes were made for both scenarios in the architecture or hyperparameters.

TABLE I. Hyperparameters of PINN-2 Experiment A.

Hyperparameters		Description
Ne	2000	Epochs
Nn	10	Number of neurons
Ni	21	Number of points in the domain [0,1].
lr	0.005	Learning rate
Tb	600	Training batches

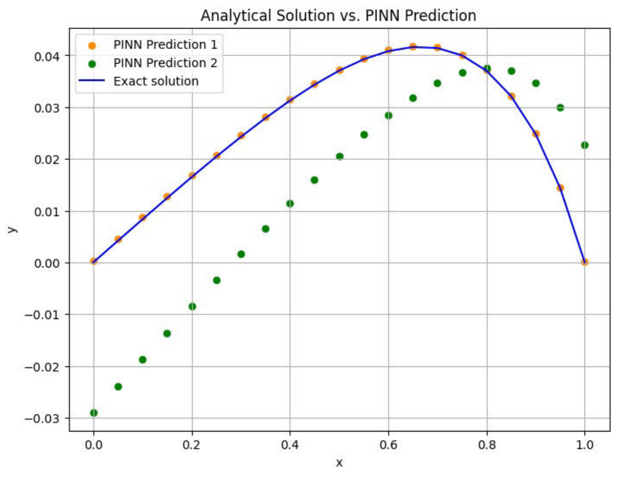

FIGURE 4. PINN Prediction 1 (orange) and PINN Prediction 2 (green) are compared to the analytical solution (blue) using the parameters from Table I.

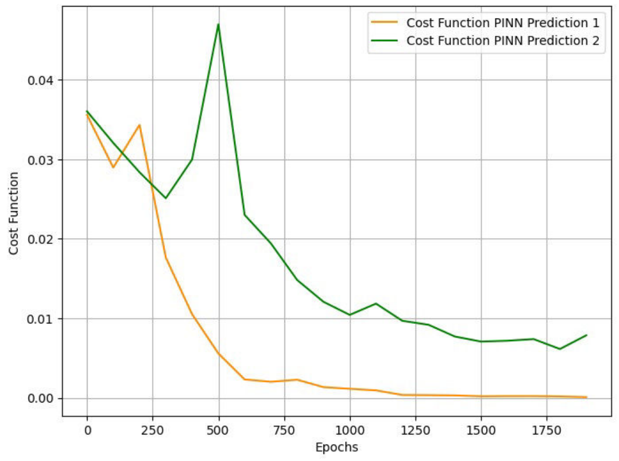

FIGURE 5. The convergence of the cost function for PINN Prediction 1 (orange) and PINN Prediction 2 (green).

Every time we run the PINN, we obtain a different solution, for the next experiments, we perform a statistical analysis, in which, we repeat the experiment 100 times and we compute the mean and standard deviations of the solutions obtained with these repetitions. For those cases, the best solution is the one with lower mean and smaller standard deviation.

3.2. Experiment B

In this experiment, the hyperparameters learning rate lr and training batches Tb were modified, as described in Table II. Each value of Tb is paired with all proposed values of lr. For example, when $Tb = 300$, it is combined with lr values of 0.005, 0.001, 0.05, and 0.01, and so on. This gives us a total

TABLE II. Mean of the minima of the Cost Function value (MMCF) and Standard Deviation for different lr and Tb values.

lr	Tb	MMCF	
		Value	Standard Deviation
0.005	300	0.0502	0.1858
	600	0.1153	0.3082
	900	0.0609	0.2353
	1200	0.0131	0.0950
0.001	300	0.0240	0.0031
	600	0.0222	0.0079
	900	0.0142	0.0073
	1200	0.0268	0.0983
0.05	300	0.0000	0.0000
	600	0.5488	0.4802
	900	5243.2396	52166.0617
	1200	502295852.0	4997780620.0
0.01	300	0.0925	0.2773
	600	0.1634	0.3616
	900	0.0872	0.2757
	1200	0.0663	0.2354

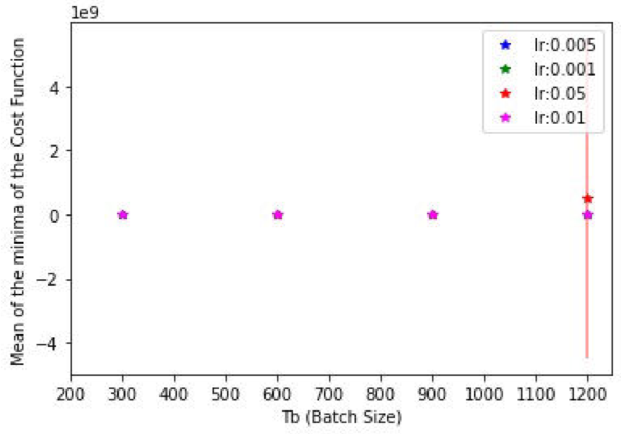

FIGURE 6. Tb (Batch size) vs Mean of the minima of the Cost Function for the 16 combinations.

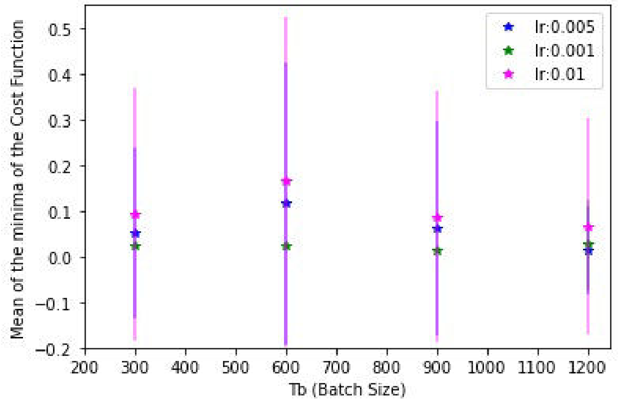

FIGURE 7. Tb (Batch size) vs Mean of the minima of the Cost Function, excluding $lr = 0.05$ data due to the lack of a discernible minimum, given the large standard deviation as shown in Table II.

of 16 possible combinations. The primary objective was to find optimal hyperparameters while minimizing the loss function. Table II provides the values of Tb, lr, their corresponding Mean of the Minima of the Cost Function (MMCF), and their respective standard deviations.

As mentioned before, we repeat each experiment 100 times, for each repetition we get the cost function for the 16 possible combinations. From each cost function, we take the minimum value (assuming that the last value of the function is the minimum), and from these 100 values, we calculate the mean (MMCF) and plot it for each Tb and lr, as shown in Fig. 6.

Throughout the process, the hyperparameters $Ne = 2000$, $Nn = 10$, and $Ni = 21$ remained constant.

In Fig. 6, a very high standard deviation is observed for $Tb = 1200$. For a better visualization, we present Fig. 7, which provides a zoom into the area of interest.

From Figs. 6 and 7, it is observed that the best performance of the PINN is achieved with the hyperparameter values $lr = 0.001$ and $Tb = 300$ (green stars). Table II confirms this minimum value, also highlighting its low standard deviation. Furthermore, this observation is visually validated in Fig. 7, where the green stars corresponding to $lr = 0.001$ are closer to zero compared to the other blue stars representing $lr = 0.005$, as well as the pink stars corresponding to $lr = 0.01$, across values of Tb.

Therefore, in the subsequent experiments, we compare the PINN with the hyperparameters $lr = 0.001$ and $Tb = 300$, referred to as PINN-1, against the values $lr = 0.005$ and $Tb = 600$, reported in the article by Andreas Almqvist (2021) [38], referred to as PINN-2.

3.3. Experiment C

In this experiment, we modified the hyperparameter Ni that represents the number of grid points for the solution domain [0,1], randomly selected from the space. We selected the value $N_i = 21$, as in the reference article, and included a higher value, $N_i = 30$, as well as a lower value, $N_i = 10$. Then, we compared PINN-1 (with the optimal hyperparameters $lr = 0.001$ and $Tb = 300$ found in Experiment 2) with PINN-2 (using the hyperparameters from the article by Andreas Almqvist (2021) [38], as shown in Table III), and with the Finite Differences method (FD).

In this experiment, we calculated the RMSE (defined as the square root of the MSE previously defined in Eq. (8)) between the analytical solution and the one predicted by PINN1 and PINN2 in each repetition (out of the 100 available). This provides us with 100 RMSE values for each network. Subsequently, we computed the Mean of these 100 RMSE values (MRMSE) along with their standard deviation, resulting in a single value for each Ni.

Figure 8 shows a comparison between two networks PINN-1 (green stars) and PINN-2 (blue stars), and the finite difference method (red stars). In Fig. 8, a significantly high standard deviation is observed for $Ni = 21$ for PINN-2. For a better visualization, we include Fig. 9, which is a zoom into the area of interest.

For the FD method, we calculate the RMSE between the analytical solution and the solution obtained by the method. The Fig. 10 illustrates the results of the Finite Differences method for different numbers of grid points ($Ni = 10, 21, 30$), compared to the analytical solution.

Once the value of Ni is set in the FD method, the calculation is performed deterministically and yields a unique and consistent result. There is no need to repeat the calculation

TABLE III. Hyperparameters of PINNs in experiment C.

Modified Hyperparameters			
Ni	10	21	30

Fixed Hyperparameters		
	PINN-1	PINN-2
Ne	2000	2000
Nn	10	10
lr	0.001	0.005
Tb	300	600

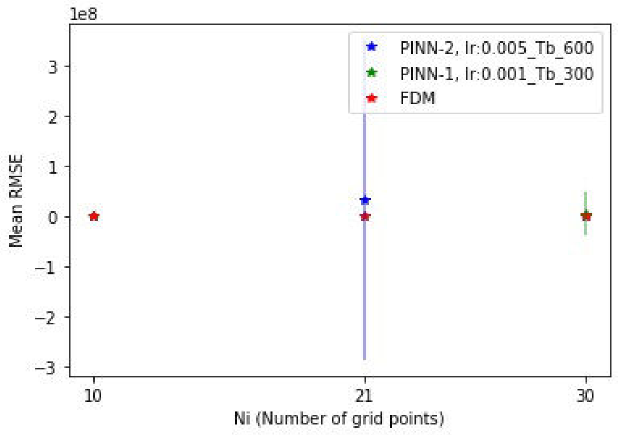

FIGURE 8. Comparison between PINN-1, PINN-2, and the FD method. On the x-axis, the number of points $Ni = 10, 21, 30$; on the y-axis, Mean RMSE (MRMSE) between the Analytical Solution and the PINN Prediction.

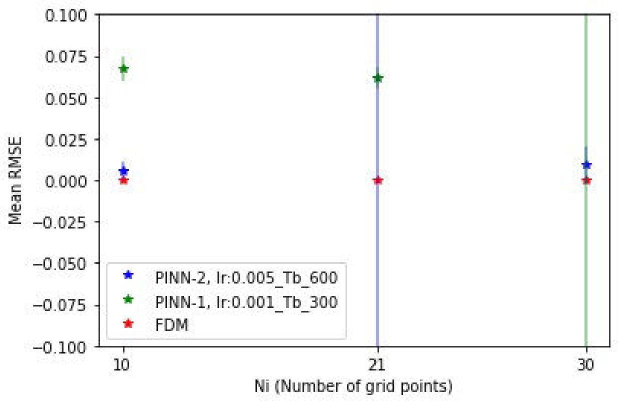

FIGURE 9. Zooming into the comparison between PINN-1, PINN-2, and the FD method. On the x-axis, the number of points $Ni = 10, 21, 30$; on the y-axis, Mean RMSE (MRMSE) between the Analytical Solution and the PINN Prediction.

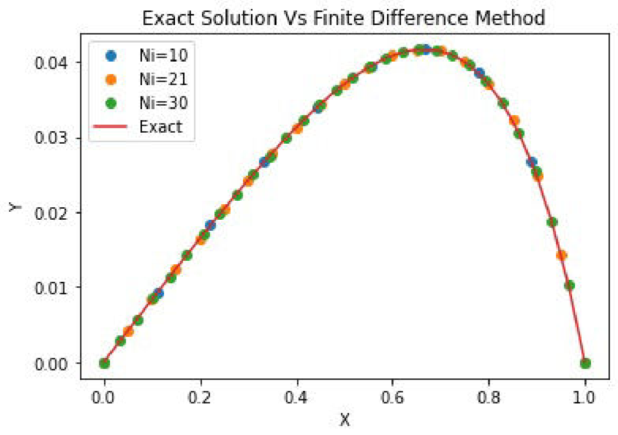

FIGURE 10. Analytical solution Vs finite difference method for $Ni = 10, 21, 30$.

TABLE IV. Mean RMSE (MRMSE) between the Analytical Solution and the prediction and Standard Deviation for PINN1, PINN-2, and FD.

	Ni	MRMSE	
		Value	Standard Deviation
PINN-1	10	0.0672	0.00717
	21	0.0615	0.00625
	30	4.307×10^6	4.284×10^7
PINN-2	10	0.005874	0.005433
	21	3.206×10^7	3.190×10^8
	30	0.008841	0.01128
FD	10	7.927×10^{-6}	0.0
	21	1.656×10^{-6}	0.0
	30	7.931×10^{-7}	0.0

multiple times, as the solution obtained is the same each time the method is applied with the same parameters and initial conditions.

This allows us to compare the accuracy of the finite differences method for different values of Ni compared to the PINNs, as shown in Table IV. The value of RMSE in the case of FD decreases as the number of points increases and is very small compared to the value of MRMSE for the PINNs in all cases. The value of MRMSE for PINN-1 and PINN-2 both exhibit a large standard deviation for $Ni = 30$ and $Ni = 21$ respectively. However, for $Ni = 10$, PINN-2 has a lower MRMSE than PINN-1.

Even if the FD method gives better results in this case, the problem we are solving is a 1D problem, for more complicated problems and inverse problems, the PINNs can be useful, especially if the problems require large computations times and they require to include data, such as in numerical reservoir simulations [31-33]. More studies are required to assess the full capacity of the PINNs.

3.4. Experiment D

In this experiment, we varied Ne, the number of epochs, from 1500, 2000, 2500, and 5000 for both PINN-1 and PINN-2.

TABLE V. Hyperparameters of the PINNs in Experiment D.

Modified Hyperparameters				
Ne	1500	2000	2500	5000

Fixed Hyperparameters		
	PINN-1	PINN-2
Ni	21	21
Nn	10	10
lr	0.001	0.005
Tb	300	600

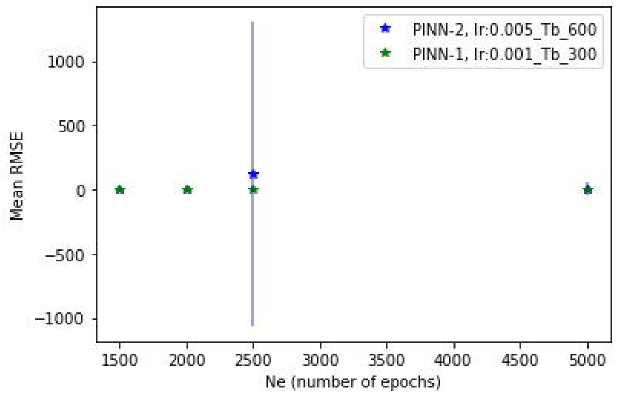

FIGURE 11. The figure illustrates the relationship between the number of epochs Ne = 1500, 2000, 2500, and 5000 and the Mean RMSE for PINN-1 (green) and PINN-2 (blue).

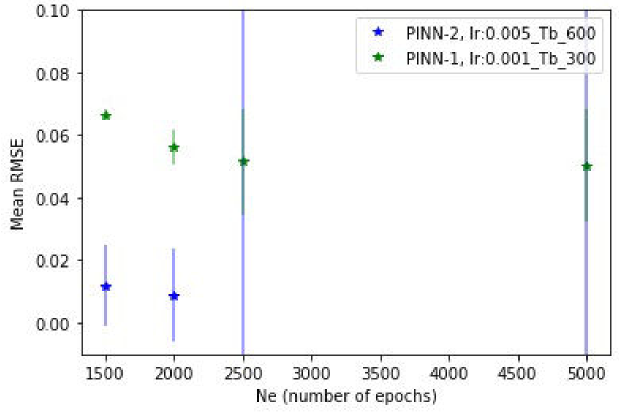

FIGURE 12. The figure provides a zoom of the relationship between the number of epochs Ne = 1500, 2000, 2500, and 5000 and the Mean RMSE for PINN-1 (green) and PINN-2 (blue).

TABLE VI. Mean RMSE (MRMSE) between the Analytical Solution and the prediction and Standard Deviation for PINN1, PINN-2.

	Ne	MRMSE	
		Value	Standard Deviation
PINN-1	1500	0.0663	0.0018
	2000	0.0563	0.0057
	2500	0.0516	0.0170
	5000	0.0503	0.0178
PINN-2	1500	0.0119	0.0128
	2000	0.0087	0.0149
	2500	118.98	1183.8
	5000	5.444	54.09

The description of the modified hyperparameters and the fixed parameters are found in Table V.

In this experiment, we calculated the RMSE between the analytical solution and the one predicted by PINN-1 and PINN-2 in each repetition (out of the 100 available), as in Experiment D. This provides us with 100 RMSE values for each value of Ne. Subsequently, we computed the mean of these 100 RMSE values (MRMSE) along with their standard deviation, resulting in a single value for each Ne, as show in Fig. 11. For a better visualization we include Fig. 12, which is a zoom into the area of interest.

In Table VI, the mean RMSE (MRMSE) between the analytical solution and the prediction and the standard deviation for PINN1, PINN-2 are presented. It can be seen that the MRMSE values between PINN-1 and PINN-2 are comparable at $Ne = 1500$ and $Ne = 2000$; for values of $Ne = 2500$ and $Ne = 5000$, PINN-1 exhibits lower MRMSE values than PINN-2.

3.5. Experiment E

In this experiment, we varied Nn the number of neurons from 5, 10, 20, and 40 for both PINN-1 and PINN-2 networks.

We compared the behavior of the optimized PINN-1 network with the reference PINN-2 network. The description of the modified hyperparameters and the fixed parameters are found in Table VII.

In this experiment, we calculated the RMSE between the analytical solution and the one predicted by PINN-1 and PINN-2 in each repetition (out of the 100 available). This provides us with 100 RMSE values for each network. Subsequently, we computed the Mean of these 100 RMSE values

TABLE VII. Hyperparameters of the PINNs in Experiment E.

Modified Hyperparameters				
Nn	5	10	20	40

Fixed Hyperparameters		
	PINN-1	PINN-2
Ne	2000	2000
Ni	21	21
lr	0.001	0.005
Tb	300	600

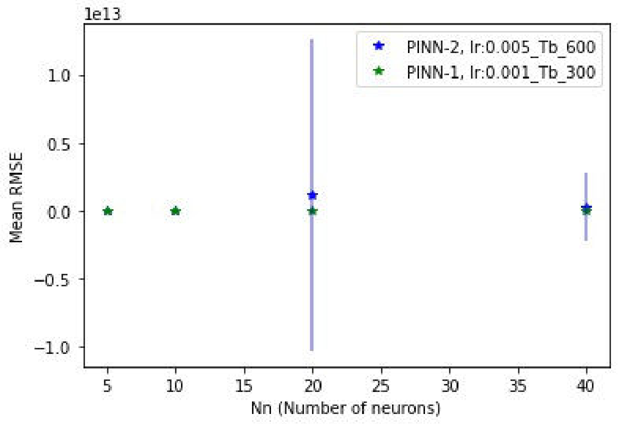

FIGURE 13. The figure illustrates the relationship between the number of neurons $Nn = 5, 10, 20,$ and 40 and the Mean RMSE for PINN-1 (green) and PINN-2 (blue).

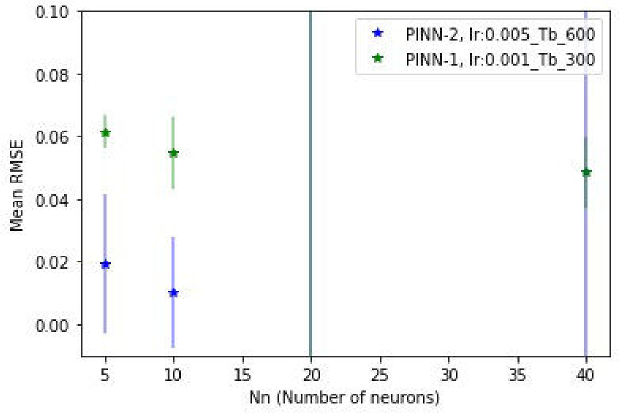

FIGURE 14. The figure provides a zoom of the relationship between the number of neurons $Nn = 5, 10, 20$, and 40 and the Mean RMSE for PINN-1 (green) and PINN-2 (blue).

TABLE VIII. Mean RMSE (MRMSE) between the Analytical Solution and the prediction and Standard Deviation for PINN-1, PINN-2.

	Nn	MRMSE	
		Value	Standard Deviation
PINN-1	5	0.0614	0.0054
	10	0.0548	0.0117
	20	23.65	181.2
	40	0.0484	0.0113
PINN-2	5	0.0193	0.0223
	10	0.0100	0.0177
	20	1.149	1.143e+13
	40	2.504e+11	2.488e+12

(MRMSE) along with their standard deviation, resulting in a single value for each Nn, as shown in Fig. 13. For a better visualization, we include Fig. 14, which is a zoom into the area of interest.

In Table VIII, the values of MRMSE and their standard deviation for each PINN are presented. It is observed that for $Nn = 5$ and 10, both PINNs show comparable values. However, in the case of $Nn = 20$, both PINN-1 and PINN-2 exhibit high MRMSE values, as well as high standard deviation, which occurs in both cases. Regarding $Nn = 40$, PINN-1 shows considerably lower MRMSE and standard deviation than PINN-2.

4. Conclusion

In this paper, we present to optimize a Physics Informed Neuran Network (PINNs) to solve the stationary one-dimensional Reynolds equation.

Within the conducted studies, a comparison is made between the solutions obtained using the PINN and the analytical solution. We performed five experiments. In Experiment A, where we replicated the experiment conducted by Almqvist *et al.* [38], and compared it with the analytical solution the PINN with this hyperparameters is referred to as PINN-2. We observe that the PINN prediction does not always correctly approximate the analytical solution of the Reynolds equation. Therefore, we decided to perform a statistical analysis by varying hyperparameters to find the optimal architecture for the PINN. Later, in Experiments B to E, we study diverse scenarios to optimize the PINN, where we vary the hyperparameters such as the learning rate (lr) and batch size (Tb), number of points (Ni), number of epochs (Ne), and number of neurons (Nn).

For Experiment B we modified the learning rate (lr) and training batches (Tb) to 0.01, 0.01, 0.05, 0.005, and 300, 600, 900, and 1200 respectively. Each value of Tb was paired with all proposed values of lr, resulting in a total of 16 possible combinations. For the statistical analysis, we repeated each experiment 100 times. In each repetition, we obtained the cost function for the 16 possible combinations. From each cost function, we selected the minimum value and calculated the mean (MMCF) and corresponding standard deviations. As a result, we have identified the optimal values for the learning rate ($lr = 0.001$) and ($Tb = 300$). These optimal values correspond to the points at which the cost function reaches its minimum and exhibit a small standard deviation. The PINN with these hyperparameters is referred to as PINN-1. From here onwards, we conduct a statistical analysis comparing PINN-1 and PINN-2. In general, for the next experiments, we observe a better performance for the PINN that we proposed (PINN-1), we elaborate next.

In Experiment C, we modified the hyperparameter Ni that represents the number of grid points for the solution domain [0,1], randomly selected from the space. We selected the value $N_i = 21$, as in the reference article, and included a higher value, $N_i = 30$, and a lower value, $N_i = 10$, and we compared PINN-1 with PINN-2 and with the Finite Differences method (FD). We calculated the RMSE between the analytical solution and the one predicted by PINN-1 and PINN-2 in each repetition (out of the 100 available), and computed the mean of these 100 RMSE values (MRMSE) along with their standard deviation. For the FD method, we calculate the RMSE between the analytical solution and the solution obtained by the method. We observe that the value of RMSE in the case of FD decreases as the number of points increases and is very small compared to the value of MRMSE for the PINNs in all cases. The value of MRMSE for PINN-1 and PINN-2 both exhibit a large standard deviation for $Ni = 30$ and $Ni = 21$ respectively. However, for $Ni = 10$, PINN-2 has a lower MRMSE than PINN-1. Even if the FD method shows better results, the PINNs are capable of approximating the solution up to $\mathcal{O}(10^{-2})$, this could be helpful for problems where using a traditional discretization scheme is complicated or when it is required to include data from the problem, that FD is not capable to do. As next steps, we plan to solve one of these problems.

For Experiment D, we varied Ne, the number of epochs, from 1500, 2000, 2500, and 5000 for both PINN-1 and PINN-2, and we calculated the RMSE between the analytical solution and the one predicted by PINN-1 and PINN-2 in each repetition (out of the 100 available). The MRMSE values between PINN-1 and PINN-2 are comparable at $Ne = 1500$ and $Ne = 2000$; for values of $Ne = 2500$ and $Ne = 5000$, PINN-1 exhibits lower MRMSE values than PINN-2.

For Experiment E, we varied Nn the number of neurons from 5, 10, 20, and 40 for both PINN-1 and PINN-2. It is observed that for $Nn = 5$ and 10, both PINNs show comparable values. However, in the case of $Nn = 20$, both PINN-1 and PINN-2 exhibit high MRMSE values, as well as high standard deviation, which occurs in both cases. Regarding $Nn = 40$, PINN-1 shows considerably lower MRMSE and standard deviation than PINN-2.

Appendix

A. Detailed description of the Neural Network elements and derivatives required for Training.

This appendix provides a detailed description of the theory presented in Subsec. 2.1 regarding the construction and computation of the PINN cost function applied to solving the Reynolds equation.

As described in Subsec. 2.1, the work involves the Reynolds equation in its dimensionless form Eq. (10), along with its boundary conditions Eq. (12) and its analytical solution Eq. (13). Subsection 2.1 also defines the problem in terms of operators Eqs. (16) and (17) and describes the cost function in its general form Eq. (19) and a specific case Eq. (22). For training the PINN, this cost function is minimized using the gradient descent algorithm, where weights and biases are iteratively updated as indicated in Eq. (25).

This appendix also expands on the theory presented in Subcec. 2.2, which discusses the neural network architecture and the operations required to compute the network output $y(x)$. Below is a detailed description of the neural network elements used in this study. We begin by introducing the nomenclature used in this appendix and throughout the article, followed by a description of the network components, including its architecture and activation function. We also provide the activation function's derivatives, which are essential for the subsequent calculation of the cost function. The cost function is then described in detail using algebraic expressions.

Nomenclature

- The superscript (j) indicates the layer number.
- The subscript i refers to the neuron number. For example, $w_i^{(0)}$ represents the weight associated with neuron i in the first layer (0).

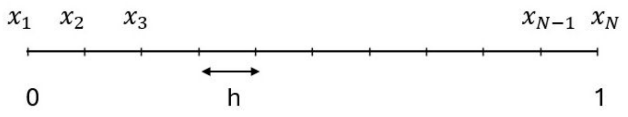

FIGURE 15. Domain divided into N points.

- Vectors (lowercase letters in boldface): such as $\boldsymbol{x}$, represent a vector with components $\boldsymbol{x} = [x_1, \ldots, x_n \ldots, x_N] \in \mathbb{R}^N$, where $x_n \in \mathbb{R}$ for $n = 1, \ldots, N$.
- $\mathbb{R}^N$ represents the N-dimensional space.

The elements of the network

To predict the output $y(\boldsymbol{x})$ of the PINN, the solution domain [0,1] is divided into N points, as illustrated in Fig. 15, creating $N-1$ equal subintervals of length h (uniformly spaced mesh points). A value x_n is randomly selected from the vector $\boldsymbol{x} = [x_1, \ldots, x_n \ldots, x_N]$, where $x_n \in [0,1]$ and $n = 1..N$. This value is used to train the network.

The network consists of an input layer where a random element x_n from the vector $\boldsymbol{x} = [x_1, \ldots, x_n \ldots, x_N] \in \mathbb{R}^N$ is taken as the input value. This value is then multiplied by all the weights of the first layer, where each $w_{i=1,\ldots,N_n}^{(0)} \in \mathbb{R}$, is an element of the weights vector $\boldsymbol{w}^{(0)} = [w_1^{(0)}, \ldots, w_i^{(0)}, \ldots w_{N_n}^{(0)}] \in \mathbb{R}^{N_n}$ where N_n is the number of neurons, thereafter, the corresponding bias $b_{i=1,\ldots,N_n}^{(0)} \in \mathbb{R}$, is added,$b_i$ is an element of the bias vector $\boldsymbol{b}^{(0)} = [b_1^{(0)}, b_2^{(0)}, \ldots, b_{N_n}^{(0)}] \in \mathbb{R}^{N_n}$.

The result of these operations is then passed through the sigmoid activation function φ, which gives us the output data for each neuron in layer (0): $z_i^{(0)}(x_n) = \varphi(w_i^{(0)} x_n + b_i^{(0)}) \in \mathbb{R}, i = 1, ..., N_n$, which are elements of the vector $\boldsymbol{z}^{(0)}(x_n) = [z_1{}^{(0)}(x_n), ..., z_i{}^{(0)}(x_n), ..., z_{N_n}^{(0)}] \in \mathbb{R}^{N_n}$ [see Fig. 16a)].

Subsequently, this result is multiplied by the second set of weights, where each $w_{i=1,\ldots,N_n}^{(1)} \in \mathbb{R}$ is an element of the weights vector $\boldsymbol{w}^{(1)} = [w_1^{(1)}, \ldots, w_i^{(1)}, \ldots w_{N_n}^{(1)}] \in \mathbb{R}^{N_n}$. The dot product between the vectors $\boldsymbol{w}^{(1)}$ and $\boldsymbol{z}^{(0)}$ results in a sum, performed from $i = 1$ to N_n, of the products of their corresponding components. A second bias term, $b^{(1)} \in \mathbb{R}$, is finallyadded, yielding the network output $y(x_n)$ for the point x_n [see Fig. 16b)]. The output is then obtained as follows:

$$y(x_n) = \sum_{i=1}^{N_n} w_i^{(1)} \varphi(w_i^{(0)} x_n + b_i^{(0)}) + b^{(1)} \in \mathbb{R}.$$

Below, we list the elements of the network:

- Input data set

$$\boldsymbol{x} = [x_1, \ldots, x_n \ldots, x_N] \in \mathbb{R}^N,$$

where N is the number of points into which the [0,1] domain was divided.

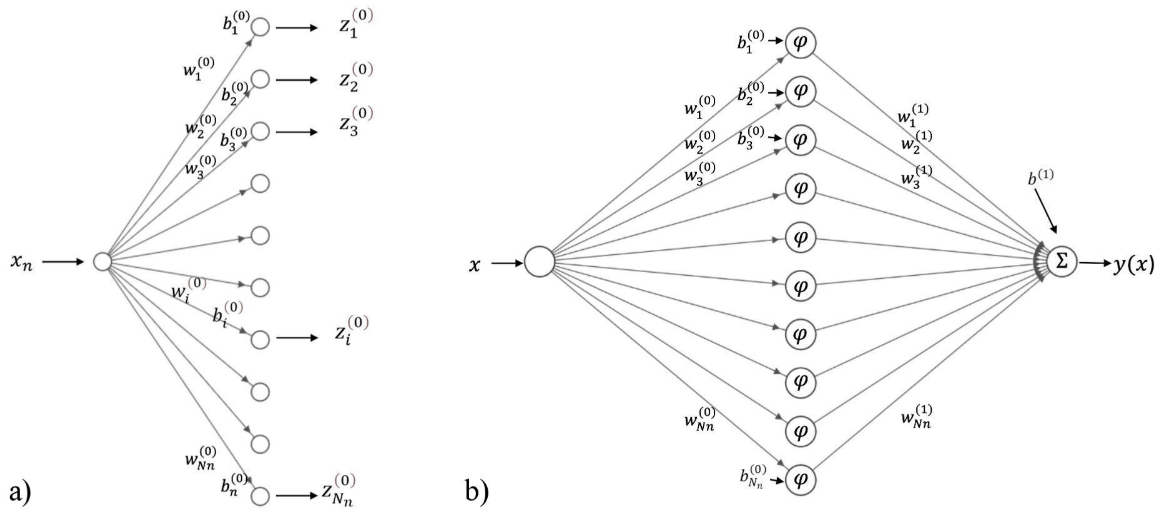

FIGURE 16. Output data for each neuron in layer (0), a) $\boldsymbol{z}^{(0)}(x_n)$. b) Output of the neural network, $y(x_n)$.

- Vector of real-valued weights for layer (0):
$$\boldsymbol{w}^{(0)} = [w_1^{(0)}, \ldots, w_i^{(0)}, \ldots, w_{N_n}^{(0)})] \in \mathbb{R}^{N_n},$$
where N_n is the number of neurons.
- Vector of real-valued biases for layer (0):
$$\boldsymbol{b}^{(0)} = [b_1^{(0)}, \ldots, b_i^{(0)}, \ldots, b_{N_n}^{(0)}] \in \mathbb{R}^{N_n}.$$
- Sigmoid activation function
$$\varphi(\varepsilon) = \frac{1}{1+e^{-\varepsilon}}, \qquad \varphi : \mathbb{R} \to [0,1].$$
- Output data for each neuron in layer (0): ${z_i}^{(0)}(x_n) = \varphi({w_i}^{(0)} x_n + {b_i}^{(0)}) \in \mathbb{R}$.
- The vector output containing all the neurons is:
$$\boldsymbol{z}^{(0)}(x_n) = [{z_1}^{(0)}, ..., {z_i}^{(0)}, ..., z_{N_n}^{(0)}] \in \mathbb{R}^{N_n}.$$
- Second weights vector:
$$\boldsymbol{w}_i^{(1)} = [w_1^{(1)}, \ldots, w_i^{(1)}, \ldots w_{N_n}^{(1)}] \in \mathbb{R}^{N_n}.$$
- Second bias term $b_i^{(1)} = b^{(1)} \in \mathbb{R}$.
- Output data for a single value x_n:
$$y(x_n) = \sum_{i=1}^{N_n} w_i^{(1)} \varphi(w_i^{(0)} x_n + b_i^{(0)}) + b^{(1)} \in \mathbb{R}.$$

Activation function

We use as activation function the sigmoid function, defined as:

$$\varphi(\varepsilon) = \frac{1}{1+e^{-\varepsilon}}, \quad \varphi : \mathbb{R} \to [0,1]. \tag{A.1}$$

For future calculations, we need its first, second, and third derivatives. Therefore, we compute them as follows:
1st Derivative of the Activation Function:

$$\varphi'(\varepsilon) = \frac{d\varphi(\varepsilon)}{d\varepsilon} = \frac{d}{d\varepsilon}\left(\frac{1}{1+e^{-\varepsilon}}\right) = \frac{d}{d\varepsilon}\left(1+e^{-\varepsilon}\right)^{-1} = -\frac{1}{(1+e^{-\varepsilon})^2} \cdot \left(-e^{-\varepsilon}\right) = \frac{e^{-\varepsilon}}{(1+e^{-\varepsilon})^2}$$
$$= \frac{1}{1+e^{-\varepsilon}} \cdot \frac{e^{-\varepsilon}}{1+e^{-\varepsilon}} = \frac{1}{1+e^{-\varepsilon}} \cdot \frac{1+e^{-\varepsilon}-1}{1+e^{-\varepsilon}} = \frac{1}{1+e^{-\varepsilon}} \cdot \frac{1+e^{-\varepsilon}}{1+e^{-\varepsilon}} - \frac{1}{1+e^{-\varepsilon}} = \varphi(\varepsilon)(1-\varphi(\varepsilon)),$$

that is:

$$\varphi'(\varepsilon) = \varphi(\varepsilon)(1-\varphi(\varepsilon)). \tag{A.2}$$

2nd Derivative of the Activation Function:

$$\begin{aligned}
\varphi''(\varepsilon) &= \frac{d^2\varphi(\varepsilon)}{d\varepsilon^2} = \frac{d}{d\varepsilon}\left[\varphi(\varepsilon)(1-\varphi(\varepsilon))\right] = \frac{d\varphi(\varepsilon)}{d\varepsilon}(1-\varphi(\varepsilon)) + \varphi(\varepsilon)\frac{d}{d\varepsilon}(1-\varphi(\varepsilon)) = \frac{d\varphi(\varepsilon)}{d\varepsilon}(1-\varphi(\varepsilon)) - \varphi(\varepsilon)\frac{d\varphi(\varepsilon)}{d\varepsilon} \\
&= \frac{d\varphi(\varepsilon)}{d\varepsilon} - \frac{d\varphi(\varepsilon)}{d\varepsilon}\varphi(\varepsilon) - \varphi(\varepsilon)\frac{d\varphi(\varepsilon)}{d\varepsilon} = \frac{d\varphi(\varepsilon)}{d\varepsilon} - 2\varphi(\varepsilon)\frac{d\varphi(\varepsilon)}{d\varepsilon} = \varphi(\varepsilon)(1-\varphi(\varepsilon)) - 2\varphi(\varepsilon)\left[\varphi(\varepsilon)(1-\varphi(\varepsilon))\right] \\
&= \varphi(\varepsilon)(1-\varphi(\varepsilon))(1-2\varphi(\varepsilon)) = \varphi'(\varepsilon)(1-2\varphi(\varepsilon)),
\end{aligned}$$

that is:

$$\varphi''(\varepsilon) = \varphi(\varepsilon)(1-\varphi(\varepsilon))(1-2\varphi(\varepsilon)) = \varphi'(\varepsilon)(1-2\varphi(\varepsilon)). \tag{A.3}$$

3rd Derivative of the Activation Function:

$$\begin{aligned}
\frac{d}{d\xi}\left(\varphi''(\xi)\right) &= \frac{d}{d\xi}\left(\varphi'(\xi)(1-2\varphi(\xi))\right) = \frac{d\varphi'(\xi)}{d\xi}(1-2\varphi(\xi)) + \varphi'(\xi)\frac{d}{d\xi}(1-2\varphi(\xi)) = \varphi''(\xi)(1-2\varphi(\xi)) + \varphi'(\xi)\cdot(-2\varphi'(\xi)) \\
&= \varphi''(\xi)(1-2\varphi(\xi)) - 2\left(\varphi'(\xi)\right)^2 = \varphi(\xi)(1-\varphi(\xi))(1-2\varphi(\xi))(1-2\varphi(\xi)) - 2\left(\varphi(\xi)(1-\varphi(\xi))\right)^2 \\
&= \varphi(\xi)(1-\varphi(\xi))\left[(1-2\varphi(\xi))^2 - 2\varphi(\xi)(1-\varphi(\xi))\right] = \varphi(\xi)(1-\varphi(\xi))\left[1-4\varphi(\xi)+4(\varphi(\xi))^2 - 2\varphi(\xi) + 2(\varphi(\xi))^2\right] \\
&= \varphi(\xi)(1-\varphi(\xi))\left[1-6\varphi(\xi)+6(\varphi(\xi))^2\right] = \varphi(\xi)(1-\varphi(\xi))[(1-\varphi(\xi))(1-3\varphi(\xi))] \\
&= \varphi(\xi)(1-\varphi(\xi))^2(1-3\varphi(\xi)),
\end{aligned}$$

that is:

$$\varphi'''(\varepsilon) = \varphi(\varepsilon)(1-\varphi(\varepsilon))^2(1-3\varphi(\varepsilon)). \tag{A.4}$$

Cost function

The cost function for the PINN, in terms of operators, in our case includes only the loss associated with the PDE and the loss due to the boundary conditions, as given by Eq. (19). It is expressed as:

$$\mathcal{L}(\boldsymbol{\theta}) = \left\langle (My-f)^2 \right\rangle + \left((Ny-\mathbf{b})\cdot\mathbf{e}_1\right)^2 + \left((Ny-\mathbf{b})\cdot\mathbf{e}_2\right)^2, \tag{A.5}$$

where $\langle Q\rangle$ is the average value of variable Q.

The cost function, substituting the residual Reynolds Eq. (20) and its boundary conditions (21), is given by:

$$\mathcal{L}(\boldsymbol{\theta}) = \left\langle (c'(x_n)y'(x_n) + c(x_n)y''(x_n) - f)^2 \right\rangle + (y(0))^2 + (y(1))^2. \tag{A.6}$$

To optimize the PINN, we need to express the cost function $\mathcal{L}(\boldsymbol{\theta})$ in terms of the weights and biases. Since the network output $y(x)$ depends on these weights and biases, we can express $\mathcal{L}(\boldsymbol{\theta})$ in terms of $y(x)$ and its derivatives as follows:

$$y(x_n) = \sum_{n=1}^{N_n} w_i^{(1)}\varphi(w_i^{(0)}x_n + b_i^{(0)}) + b^{(1)}, \tag{A.7}$$

$$y(0) = \sum_{i=1}^{N_n} {w_i}^{(1)}\varphi({b_i}^{(0)}) + b^{(1)}, \tag{A.8}$$

$$y(1) = \sum_{i=1}^{N_n} {w_i}^{(1)}\varphi({w_i}^{(0)} + {b_i}^{(0)}) + b^{(1)}, \tag{A.9}$$

$$y'(x_n) = \frac{dy}{dx} = \sum_{i=1}^{N_n} w_i^{(1)}w_i^{(0)}\varphi'(w_i^{(0)}x_n + b_i^{(0)}), \tag{A.10}$$

$$y''(x_n) = \frac{d^2y}{dx^2} = \sum_{i=1}^{N_n} w_i^{(1)}(w_i^{(0)})^2\varphi''(w_i^{(0)}x_n + b_i^{(0)}). \tag{A.11}$$

Substituting the Eqs. (A.8), (A.9), (A.10), and (A.11) into Eq. (A.6), we obtain:

$$\mathcal{L} = \left\langle \left(c'(x_n) \sum_{i=1}^{N_n} w_i^{(1)} w_i^{(0)} \varphi'(w_i^{(0)} x_n + b_i^{(0)}) + c(x_n) \sum_{i=1}^{N_n} w_i^{(1)} (w_i^{(0)})^2 \varphi''(w_i^{(0)} x_n + b_i^{(0)}) - f \right)^2 \right\rangle + \left(\sum_{i=1}^{N_n} w_i^{(1)} \varphi(b_i^{(0)}) + b^{(1)} \right)^2 + \left(\sum_{i=1}^{N_n} w_i^{(1)} \varphi(w_i^{(0)} + b_i^{(0)}) + b^{(1)} \right)^2, \tag{A.12}$$

where

$$c(x) = (1 + K - Kx)^3, \qquad c'(x) = \frac{dc(x)}{dx} = -3K(1 + K - Kx)^2.$$

Once we have the cost function expressed in terms of the neural network parameters, and since we want to minimize this function, the next step is to use an algorithm to optimize it. The algorithm we use here is Gradient Descent.

Gradient descent algorithm

The gradient descent algorithm iteratively adjusts the parameters $\boldsymbol{\theta}$, where $\boldsymbol{\theta}$ represents the set of all weights and biases across all layers of the network as previously defined in Eq. (9). In our case: $\boldsymbol{\theta} = \{\boldsymbol{w}^{(0)}, \boldsymbol{w}^{(1)}, \boldsymbol{b}^{(0)}, b^{(1)}\}$ and we want to find the set of parameters $\boldsymbol{\theta}$ that minimize the cost function $\mathcal{L}(\boldsymbol{\theta})$ as defined in Eq. (A.12).

The gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_n)$ is computed, indicating the direction of the steepest increase in the cost function. The parameters are then adjusted in the opposite direction of the gradient using:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_n),$$

where η is the learning rate, and the parameters of the neural network are initialized randomly.

This process continues until the error is less than a user-defined tolerance value or until a maximum number of iterations is reached. When either of these conditions is met, the obtained weights and biases are those that minimize the cost function within the established tolerance. The goal is to minimize the cost function value to optimize the model.

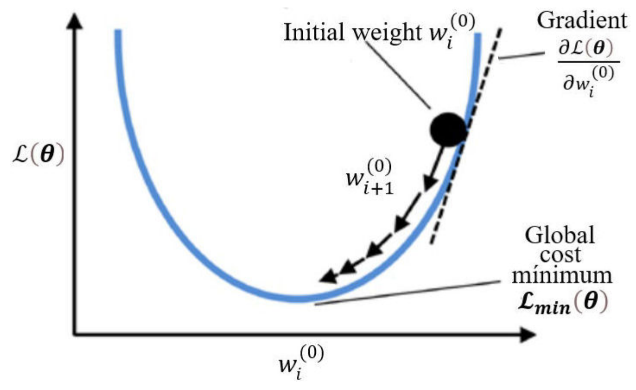
Initial weight $w_i^{(0)}$
Gradient
$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial w_i^{(0)}}$
$\mathcal{L}(\boldsymbol{\theta})$
$w_{i+1}^{(0)}$
Global cost minimum
$\mathcal{L}_{min}(\boldsymbol{\theta})$
$w_i^{(0)}$

FIGURE 17. Gradient descent algorithm.

To illustrate the process of conjugate gradient, see Fig. 17, where we have an initial parameter $w_i{}^{(0)}$. We compute the gradient of the function in this point $\partial \mathcal{L}(\boldsymbol{\theta})/\partial w_i{}^{(0)}$. The next position $w_{i+1}{}^{(0)}$ of the parameter is found taking the initial position and giving a step η in the opposite direction of the gradient (which is the direction of bigger growth of the function), *i.e.*:

$$w_{i+1}^{(0)} = w_i^{(0)} - \eta \frac{\partial \mathcal{L}\left(\boldsymbol{\theta}\right)}{\partial w_i^{(0)}}.$$

In our case, the cost function depends on the weights and biases of each layer. Therefore, the parameters are adjusted using the following formulas:

$$2w_{i+1}^{(0)} = w_i^{(0)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial w_i^{(0)}},$$
$$w_{i+1}^{(1)} = w_i^{(1)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial w_i^{(1)}},$$
$$b_{i+1}^{(0)} = b_i^{(0)} - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial b_i^{(0)}},$$
$$b_{i+1}^{(1)} = b_i^{(1)} - \eta \frac{\partial \mathcal{L}(\theta)}{\partial b_i^{(1)}}, \tag{A.13}$$

where the step η is an hyperparameter known as the learning rate.

The derivatives of the Eqs. (A.13) are computed analytically by differentiating the equations, which requires the partial derivatives of the general cost function with respect to each weight and bias. These derivatives are presented next:

$$\frac{\partial \mathcal{L}}{\partial w_i^{(0)}} = \frac{\partial \langle (My-f)^2\rangle}{\partial w_i^{(0)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_1)^2}{\partial w_i^{(0)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_2)^2}{\partial w_i^{(0)}} = \frac{\partial \langle (c'(x)y'+c(x)y''-f)^2\rangle}{\partial w_i^{(0)}}$$
$$+ \frac{\partial ((y(0)-0))^2}{\partial w_i^{(0)}} + \frac{\partial ((y(1)-0))^2}{\partial w_i^{(0)}} = \left\langle 2\left(c(x)y''+c'(x)y'-f\right)\left(c(x)\frac{\partial y''}{\partial w_i^{(0)}} + c'(x)\frac{\partial y'}{\partial w_i^{(0)}} - \frac{\partial f}{\partial w_i^{(0)}}\right)\right\rangle$$
$$+ 2(y(0)-0)\frac{\partial y(0)}{\partial w_i^{(0)}} + 2(y(1)-0)\frac{\partial y(1)}{\partial w_i^{(0)}}, \tag{A.14}$$

$$\frac{\partial \mathcal{L}}{\partial w_i^{(1)}} = \frac{\partial \langle (My-f)^2\rangle}{\partial w_i^{(1)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_1)^2}{\partial w_i^{(1)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_2)^2}{\partial w_i^{(1)}} = \frac{\partial \langle (c'(x)y'+c(x)y''-f)^2\rangle}{\partial w_i^{(1)}}$$
$$+ \frac{\partial ((y(0)-0))^2}{\partial w_i^{(1)}} + \frac{\partial ((y(1)-0))^2}{\partial w_i^{(1)}} = \left\langle 2\left(c(x)y''+c'(x)y'-f\right)\left(c(x)\frac{\partial y''}{\partial w_i^{(1)}} + c'(x)\frac{\partial y'}{\partial w_i^{(1)}} - \frac{\partial f}{\partial w_i^{(1)}}\right)\right\rangle$$
$$+ 2(y(0)-0)\frac{\partial y(0)}{\partial w_i^{(1)}} + 2(y(1)-0)\frac{\partial y(1)}{\partial w_i^{(1)}}, \tag{A.15}$$

$$\frac{\partial \mathcal{L}}{\partial b_i^{(0)}} = \frac{\partial \langle (My-f)^2\rangle}{\partial b_i^{(0)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_1)^2}{\partial b_i^{(0)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_2)^2}{\partial b_i^{(0)}} = \frac{\partial \langle (c'(x)y'+c(x)y''-f)^2\rangle}{\partial b_i^{(0)}}$$
$$+ \frac{\partial ((y(0)-0))^2}{\partial b_i^{(0)}} + \frac{\partial ((y(1)-0))^2}{\partial b_i^{(0)}} = \left\langle 2\left(c(x)y''+c'(x)y'-f\right)\left(c(x)\frac{\partial y''}{\partial b_i^{(0)}} + c'(x)\frac{\partial y'}{\partial b_i^{(0)}} - \frac{\partial f}{\partial b_i^{(0)}}\right)\right\rangle$$
$$+ 2(y(0)-0)\frac{\partial y(0)}{\partial b_i^{(0)}} + 2(y(1)-0)\frac{\partial y(1)}{\partial b_i^{(0)}}, \tag{A.16}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(1)}} = \frac{\partial \langle (My-f)^2\rangle}{\partial b^{(1)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_1)^2}{\partial b^{(1)}} + \frac{\partial ((Ny-\mathbf{b})\cdot \mathbf{e}_2)^2}{\partial b^{(1)}} = \frac{\partial \langle (c'(x)y'+c(x)y''-f)^2\rangle}{\partial b^{(1)}}$$
$$+ \frac{\partial ((y(0)-0))^2}{\partial b^{(1)}} + \frac{\partial ((y(1)-0))^2}{\partial b^{(1)}} = \left\langle 2\left(c(x)y''+c'(x)y'-f\right)\left(c(x)\frac{\partial y''}{\partial b^{(1)}} + c'(x)\frac{\partial y'}{\partial b^{(1)}} - \frac{\partial f}{\partial b^{(1)}}\right)\right\rangle$$
$$+ 2(y(0)-0)\frac{\partial y(0)}{\partial b^{(1)}} + 2(y(1)-0)\frac{\partial y(1)}{\partial b^{(1)}}. \tag{A.16}$$

The expressions (A.14), (A.15), (A.16), and (A.16), require a set of derivatives, which we calculate below.
For Eq. (A.14), we need,

$$\frac{\partial y'}{\partial w_i^{(0)}}, \quad \frac{\partial y''}{\partial w_i^{(0)}}, \quad \frac{\partial y(0)}{\partial w_i^{(0)}}, \quad \frac{\partial y(1)}{\partial w_i^{(0)}}.$$

From Eqs. (A.7), (A.8), (A.9), (A.10), and (A.11), we have the values of $y(x)$, $y(0)$, $y(1)$, y', and y'' respectively. Here, we will need the activation function and its respective derivatives, which are given by Eqs. (A.1), (A.2), (A.3), and (A.4). Substituting the values of y', y'', $y(0)$, and $y(1)$ and differentiating with respect to $w_i^{(0)}$, we obtain the following:

$$\frac{\partial y'}{\partial w_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)}\varphi'(w_i^{(0)}x_n + b_i^{(0)}) + \sum_{i=1}^{N_n} x_n w_i^{(1)}(w_i^{(0)})^2\varphi''(w_i^{(0)}x_n + b_i^{(0)}), \tag{A.17}$$

$$\frac{\partial y''}{\partial w_i^{(0)}} = \sum_{i=1}^{N_n} 2w_i^{(1)}w_i^{(0)}\varphi''(w_i^{(0)}x_n + b_i^{(0)}) + \sum_{i=1}^{N_n} x_n w_i^{(1)}(w_i^{(0)})^2\varphi'''(w_i^{(0)}x_n + b_i^{(0)}), \tag{A.18}$$

since

$$y(0) = \sum_{i=1}^{N_n} w_i^{(1)}\varphi(w_i^{(0)}(0) + b_i^{(0)}) + b^{(1)} = \sum_{i=1}^{N_n} w_i^{(1)}\varphi(b_i^{(0)}) + b^{(1)}, \tag{A.19}$$

then

$$\frac{\partial y(0)}{\partial w_i^{(0)}} = 0, \tag{A.20}$$

since

$$y(1) = \sum_{i=1}^{N_n} w_i^{(1)} \varphi(w_i^{(0)}(1) + b_i^{(0)}) + b^{(1)}, \tag{A.21}$$

then

$$\frac{\partial y(1)}{\partial w_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(w_i^{(0)}(1) + b_i^{(0)})(1) = \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(w_i^{(0)} + b_i^{(0)}). \tag{A.21}$$

For Eq. (A.15), we need:

$$\frac{\partial y'}{\partial w_i^{(1)}}, \quad \frac{\partial y''}{\partial w_i^{(1)}}, \quad \frac{\partial y(0)}{\partial w_i^{(1)}}, \quad \frac{\partial y(1)}{\partial w_i^{(1)}}.$$

From Eqs. (A.7), (A.8), (A.9), (A.10), and (A.11), we have the values of $y(x)$, $y(0)$, $y(1)$, y', and y'' respectively. Here, we will need the activation function and its respective derivatives, which are given by Eqs. (A.1), (A.2), (A.3), and (A.4).

Substituting the values of y', y'', $y(0)$, and $y(1)$ and differentiating with respect to $w_i^{(1)}$, we obtain the following:

$$\frac{\partial y'}{\partial w_i^{(1)}} = \sum_{i=1}^{N_n} w_i^{(0)} \varphi'(w_i^{(0)} x_n + b_i^{(0)}), \tag{A.22}$$

$$\frac{\partial y''}{\partial w_i^{(1)}} = \sum_{i=1}^{N_n} (w_i^{(0)})^2 \varphi''(w_i^{(0)} x_n + b_i^{(0)}), \tag{A.23}$$

since

$$\frac{\partial y(x)}{\partial w_i^{(1)}} = \sum_{i=1}^{N_n} \varphi(w_i^{(0)} x_n + b_i^{(0)}), \tag{A.24}$$

then:

$$\frac{\partial y(0)}{\partial w_i^{(1)}} = \sum_{i=1}^{N_n} \varphi(w_i^{(0)}(0) + b_i^{(0)}) = \sum_{i=1}^{N_n} \varphi(b_i^{(0)}), \tag{A.25}$$

$$\frac{\partial y(1)}{\partial w_i^{(1)}} = \sum_{i=1}^{N_n} \varphi(w_i^{(0)}(1) + b_i^{(0)}) = \sum_{i=1}^{N_n} \varphi(w_i^{(0)} + b_i^{(0)}), \tag{A.26}$$

For Eq. (A.16), we need:

$$\frac{\partial y'}{\partial b_i^{(0)}}, \quad \frac{\partial y''}{\partial b_i^{(0)}}, \quad \frac{\partial y(0)}{\partial b_i^{(0)}}, \quad \frac{\partial y(1)}{\partial b_i^{(0)}}.$$

From Eqs. (A.7), (A.8), (A.9), (A.10), and (A.11), we have the values of $y(x)$, $y(0)$, $y(1)$, y', and y'' respectively. Here, we will need the activation function and its respective derivatives, which are given by Eqs. (A.1), (A.2), (A.3), and (A.4).

Substituting the values of y', y'', $y(0)$, and $y(1)$ and differentiating with respect to $b_i^{(0)}$, we obtain the following:

$$\frac{\partial y'}{\partial b_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)} w_i^{(0)} \varphi''(w_i^{(0)} x_n + b_i^{(0)}), \tag{A.27}$$

$$\frac{\partial y''}{\partial b_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)} (w_i^{(0)})^2 \varphi'''(w_i^{(0)} x_n + b_i^{(0)}), \tag{A.28}$$

since

$$\frac{\partial y(x)}{\partial b_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(w_i^{(0)} x_n + b_i^{(0)}), \tag{A.29}$$

then

$$\frac{\partial y(0)}{\partial b_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(w_i^{(0)}(0) + b_i^{(0)})$$
$$= \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(b_i^{(0)}), \tag{A.30}$$

$$\frac{\partial y(1)}{\partial b_i^{(0)}} = \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(w_i^{(0)}(1) + b_i^{(0)})$$
$$= \sum_{i=1}^{N_n} w_i^{(1)} \varphi'(w_i^{(0)} + b_i^{(0)}). \tag{A.31}$$

For Eq. (A.16), we need:

$$\frac{\partial y'}{\partial b^{(1)}}, \quad \frac{\partial y''}{\partial b^{(1)}}, \quad \frac{\partial y(0)}{\partial b^{(1)}}, \quad \frac{\partial y(1)}{\partial b^{(1)}}.$$

From Eqs. (A.7), (A.8), (A.9), (A.10), and (A.11), we have the values of $y(x)$, $y(0)$, $y(1)$, y', and y'' respectively. Here, we will need the activation function and its respective derivatives, which are given by Eqs. (A.1), (A.2), (A.3), and (A.4).

Substituting the values of y', y'', $y(0)$, and $y(1)$ and differentiating with respect to $b^{(1)}$, we obtain the following:

$$\frac{\partial y'}{\partial b^{(1)}} = 0, \tag{A.32}$$

$$\frac{\partial y''}{\partial b^{(1)}} = 0, \tag{A.33}$$

since

$$\frac{\partial y(x)}{\partial b^{(1)}} = 1, \tag{A.34}$$

then:

$$\frac{\partial y(0)}{\partial b^{(1)}} = 1, \tag{A.35}$$

$$\frac{\partial y(1)}{\partial b^{(1)}} = 1. \tag{A.36}$$

Once the necessary derivatives for Eqs. (A.16), (A.16), (A.17), and (A.18) have been computed, we can iteratively update the weights and biases according to Eq. (A.15). Then, we calculate the cost function using these updated weights and biases, as indicated in Eq. (A.12). The optimization is performed over T_b batches, and the entire process is repeated across the specified number of epochs.

1. Y. Chen *et al.*, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, *Optics express* **28** (2020) 11618.
2. M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561 (2017).
3. M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations, arXiv preprint arXiv:1711.10566 (2017).
4. 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* **378** (2019) 686.
5. A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, *Computer Methods in Applied Mechanics and Engineering* **365** (2020) 113028.
6. X. Jin *et al.*, NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, *Journal of Computational Physics* **426** (2021) 109951.
7. T. T. Garipov *et al.*, Unified thermo-compositional-mechanical framework for reservoir simulation, *Computational Geosciences* **22** (2018) 1039.
8. J.-L. Wu, H. Xiao, and E. Paterson, Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework, *Physical Review Fluids* **3** (2018) 074602.
9. U. binWaheed *et al.*, PINNeik: Eikonal solution using physics-informed neural networks, *Computers & Geosciences* **155** (2021) 104833.
10. C. Rao, H. Sun, and Y. Liu, Physics-informed deep learning for incompressible laminar flows, *Theoretical and Applied Mechanics Letters* **10** (2020) 207.
11. Z. Mao, A. D. Jagtap, and G. E. Karniadakis, Physics-informed neural networks for high-speed flows, *Computer Methods in Applied Mechanics and Engineering* **360** (2020) 112789.
12. B. Reyes *et al.*, Learning unknown physics of non-Newtonian fluids, *Physical Review Fluids* **6** (2021) 073301.
13. E. Haghighat *et al.*, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, *Computer Methods in Applied Mechanics and Engineering* **379** (2021) 113741.
14. C. Rao, H. Sun, and Y. Liu, Physics-informed deep learning for computational elastodynamics without labeled data, *Journal of Engineering Mechanics* **147** (2021) 04021043.
15. E. Haghighat *et al.*, A nonlocal physics-informed deep learning framework using the peridynamic differential operator, *Com-*

puter Methods in Applied Mechanics and Engineering **385** (2021) 114012.

16. M. Guo and E. Haghighat, Energy-based error bound of physics-informed neural network solutions in elasticity, *Journal of Engineering Mechanics* **148** (2022) 04022038.
17. S. Cai *et al*., Physics-informed neural networks for heat transfer problems, *Journal of Heat Transfer* **143** (2021) 060801.
18. S. A. Niaki *et al*., Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture, *Computer Methods in Applied Mechanics and Engineering* **384** (2021) 113959.
19. N. Zobeiry and K. D. Humfeld, A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications, *Engineering Applications of Artificial Intelligence* **101** (2021) 104232.
20. C. Song, T. Alkhalifah, and U. B. Waheed, Solving the frequency-domain acoustic VTI wave equation using physicsinformed neural networks, *Geophysical Journal International* **225** (2021) 846.
21. C. Song and T. A. Alkhalifah, Wavefield reconstruction inversion via physics-informed neural networks, *IEEE Transactions on Geoscience and Remote Sensing* **60** (2021) 1.
22. U. B. Waheed *et al.*, A holistic approach to computing first-arrival traveltimes using neural networks, arXiv preprint arXiv:2101.11840 (2021).
23. O. Fuks and H. A. Tchelepi, Limitations of physics informed machine learning for nonlinear two-phase transport in porous media, *Journal of Machine Learning for Modeling and Computing* **1** (2020).
24. M. M. Almajid and M. O. Abu-Al-Saud, Prediction of porous media fluid flow using physics informed neural networks, *Journal of Petroleum Science and Engineering* **208** (2022) 109205.
25. Y. W. Bekele, Physics-informed deep learning for onedimensional consolidation, *Journal of Rock Mechanics and Geotechnical Engineering* **13** (2021) 420.
26. P. Shokouhi *et al*., Physics-informed deep learning for prediction of CO2 storage site response, *Journal of Contaminant Hydrology* **241** (2021) 103835.
27. T. Kadeethum, T. M. Jørgensen, and H. M. Nick, Physicsinformed neural networks for solving nonlinear diffusivity and Biot's equations, *PloS one* **15** (2020) e0232683.
28. Y. W. Bekele, Physics-informed deep learning for flow and deformation in poroelastic media, arXiv preprint arXiv:2010.15426 (2020).
29. E. Haghighat, D. Amini, and R. Juanes, Physics-informed neural network simulation of multiphase poroelasticity using stress-split sequential training, *Computer Methods in Applied Mechanics and Engineering* **397** (2022) 115141.
30. C. Yang, X. Yang, and X. Xiao, Data-driven projection method in fluid simulation, *Computer Animation and Virtual Worlds* **27** (2016) 415.
31. Y. Zhu and N. Zabaras, Bayesian deep convolutional encoderdecoder networks for surrogate modeling and uncertainty quantification, *Journal of Computational Physics* **366** (2018) 415.
32. R. K. Tripathy and I. Bilionis, Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification, *Journal of computational physics* **375** (2018) 565.
33. S. Mo *et al*., Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media, *Water Resources Research* **55** (2019) 703.
34. J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, *Journal of Fluid Mechanics* **807** (2016) 155.
35. N. Thuerey *et al.*, Deep learning methods for Reynoldsaveraged Navier-Stokes simulations of airfoil flows, *AIAA Journal* **58** (2020) 25.
36. N. Geneva and N. Zabaras, Quantifying model form uncertainty in Reynolds-averaged turbulence models with Bayesian deep neural networks, *Journal of Computational Physics* **383** (2019) 125.
37. Y. Zhu *et al*., Physics-constrained deep learning for highdimensional surrogate modeling and uncertainty quantification without labeled data, *Journal of Computational Physics* **394** (2019) 56.
38. G. B. Diaz-Cortes and R. Luna-Garcia, A Novel Evolutionary Algorithm: One-Dimensional Subspaces Optimization Algorithm (1D-SOA), *Symmetry* **15** (2023) 1873, `https://doi.org/10.3390/sym15101873`.
39. H. Alibrahim and S. A. Ludwig, *Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization*, In 2021 IEEE Congress on Evolutionary Computation (CEC) (2021) pp. 1551-1559, `https://doi.org/10.1109/CEC45853.2021.9504761`.
40. D. Simon, Evolutionary Optimization Algorithms (John Wiley & Sons, 2013).
41. C. Coello, D. Van Veldhuizen, and G. Lamont, Evolutionary Algorithms for Solving Multi-Objective Problems, *Genetic Algorithms and Evolutionary Computation* (Springer US, 2013), `https://books.google.com.mx/books?id=VmnTBwAAQBAJ`.
42. S. Cai *et al*., Physics-informed neural networks (PINNs) for fluid mechanics: A review, *Acta Mechanica Sinica* **37** (2021) 1727.
43. A. Almqvist and F. P. Ràfols, Scientific computing with applications in tribology: A course compendium (2022).
44. D. C. Psichogios and L. H. Ungar, A hybrid neural networkfirst principles approach to process modeling, *AIChE Journal* **38** (1992) 1499.
45. I. E. Lagaris, A. Likas, and D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, *IEEE transactions on neural networks* **9** (1998) 987.
46. G. Cybenko, Mathematics of control, Signals and Systems **2** (1989) 303.
47. K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural networks* **2** (1989) 359.
48. G. Pilania *et al*., Physics-informed machine learning for inorganic scintillator discovery, *The Journal of chemical physics* **148** (2018).

49. S. Liu *et al.*, Physics-informed machine learning for composition-process-property design: Shape memory alloy demonstration, *Applied Materials Today* **22** (2021) 100898.

50. W. Ji *et al.*, Stiff-pinn: Physics-informed neural network for stiff chemical kinetics, *The Journal of Physical Chemistry A* **125** (2021) 8098.

51. Z. Fang and J. Zhan, Deep physical informed neural networks for metamaterial design, *Ieee Access* **8** (2019) 24506.

52. O. Noakoasteen, *et al.*, Physics-informed deep neural networks for transient electromagnetic analysis, *IEEE Open Journal of Antennas and Propagation* **1** (2020) 404.

53. S. Barry and G. Mercer, Exact solutions for two-dimensional time-dependent flow and deformation within a poroelastic medium, *Journal of applied mechanics* **66** (1999) 536.